	A	B	C	D	E	F	G	H	I	J
1	name	year	selling_price	km_driven	fuel	mileage	engine	max_power	torque	seats
2	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5
3	Skoda Rapid 1.5 TDI Ambitic	2014	370000	120000	Diesel	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5
4	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5
5	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5
6	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5
7	Hyundai Xcent 1.2 VTVT E Pl	2017	440000	45000	Petrol	20.14 kmpl	1197 CC	81.86 bhp	113.75nm@ 4000rpm	5
8	Maruti Wagon R LXI DUO BS	2007	96000	175000	LPG	17.3 km/kg	1061 CC	57.5 bhp	7.8@ 4,500(kgm@ rpm)	5
9	Maruti 800 DX BSII	2001	45000	5000	Petrol	16.1 kmpl	796 CC	37 bhp	59Nm@ 2500rpm	4
10	Toyota Etios VXD	2011	350000	90000	Diesel	23.59 kmpl	1364 CC	67.1 bhp	170Nm@ 1800-2400rpm	5
11	Ford Figo Diesel Celebration	2013	200000	169000	Diesel	20.0 kmpl	1399 CC	68.1 bhp	160Nm@ 2000rpm	5
12	Renault Duster 110PS Diese	2014	500000	68000	Diesel	19.01 kmpl	1461 CC	108.45 bhp	248Nm@ 2250rpm	5
13	Maruti Zen LX	2005	92000	100000	Petrol	17.3 kmpl	993 CC	60 bhp	78Nm@ 4500rpm	5
14	Maruti Swift Dzire VDi	2009	280000	140000	Diesel	19.3 kmpl	1248 CC	73.9 bhp	190Nm@ 2000rpm	5

I will be working on this dataset. I will model selling price!

In [99]:
```
import pandas as pd
df = pd.read_csv("cars.csv")
```

In [ ]:

I am normalizing year values so that it doens't have large influence.

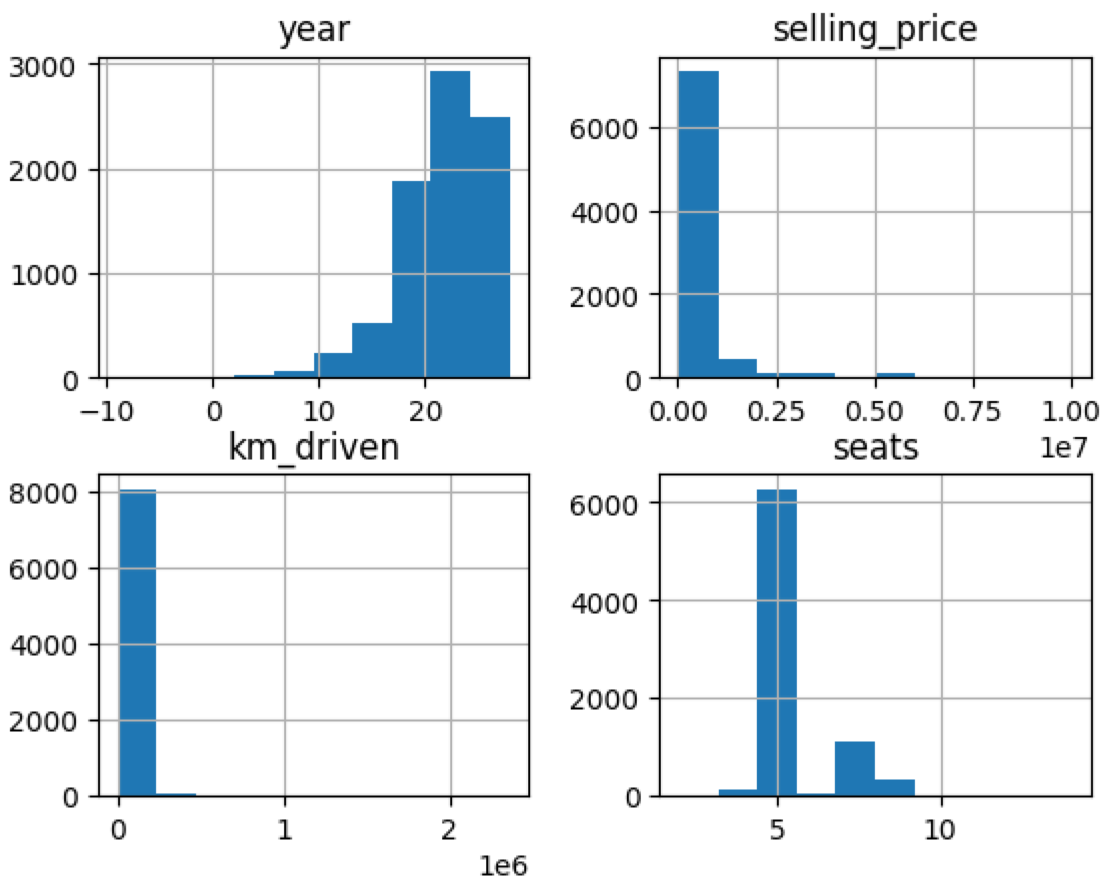
In [100…
```
df.year = df.year -1992
df.head(5)
```

Out[100…

	name	year	selling_price	km_driven	fuel	mileage	engine	max_power	torque	seats
0	Maruti Swift Dzire VDI	22	450000	145500	Diesel	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda Rapid 1.5 TDI Ambition	22	370000	120000	Diesel	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda City 2017-2020 EXi	14	158000	140000	Petrol	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai i20 Sportz Diesel	18	225000	127000	Diesel	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0
4	Maruti Swift VXI BSIII	15	130000	120000	Petrol	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0

In [101…
```
df.hist()
```

Out[101…
```
array([[<Axes: title={'center': 'year'}>,
        <Axes: title={'center': 'selling_price'}>],
       [<Axes: title={'center': 'km_driven'}>,
        <Axes: title={'center': 'seats'}>]], dtype=object)
```



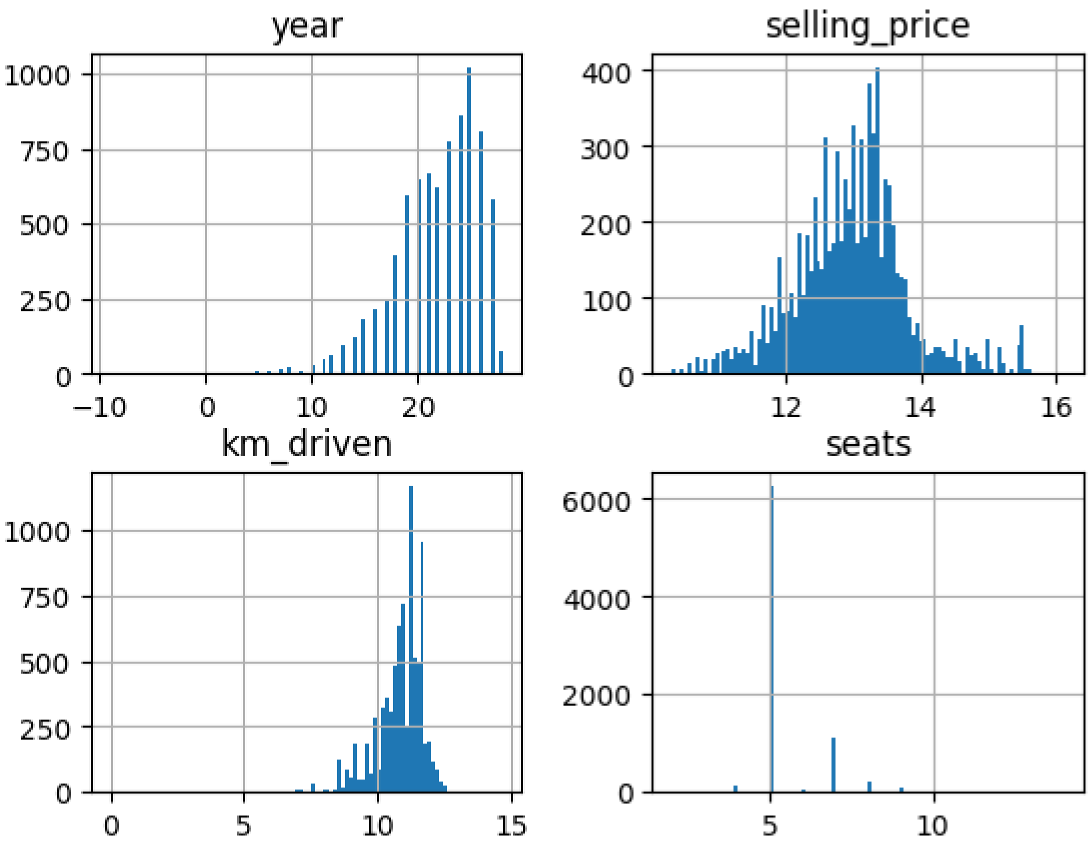
I will have to apply transformaton to Selling price and Km_driven columns because there are outliers

In [102…
```
df.km_driven = np.log(df.km_driven)
df.selling_price = np.log(df.selling_price)
```

In [103…
```
df.hist(bins = 100)
```

Out[103…

```
array([[<Axes: title={'center': 'year'}>,
        <Axes: title={'center': 'selling_price'}>],
       [<Axes: title={'center': 'km_driven'}>,
        <Axes: title={'center': 'seats'}>]], dtype=object)
```



In [104…

```
# Assuming df is your DataFrame
percentiles = [0.025, 0.975]  # Define the percentiles you want to display
df_summary = df.describe(percentiles=percentiles)
df_summary
```

Out[104…

	year	selling_price	km_driven	seats
count	8128.000000	8128.000000	8128.000000	7907.000000
mean	21.804011	12.973409	10.860092	5.416719
std	4.044249	0.839134	0.875581	0.959588
min	-9.000000	10.308919	0.000000	2.000000
2.5%	12.000000	11.289782	8.634265	5.000000
50%	23.000000	13.017003	11.002100	5.000000
97.5%	27.000000	14.978661	12.128111	8.000000
max	28.000000	16.118096	14.674366	14.000000

You can see above that minimum for year is -9 while 2.5percentile is 12. That makes me want to remove outliers

In [105…

```
lower_bound = 12.000000
upper_bound = 27.000000

# Filter for values within the specified range
df = df[(df['year'] >= lower_bound) & (df['year'] <= upper_bound)]

# Print the filtered DataFrame
df.head()
```

Out[105…

	name	year	selling_price	km_driven	fuel	mileage	engine	max_power	torque	seats
0	Maruti Swift Dzire VDI	22	13.017003	11.887931	Diesel	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda Rapid 1.5 TDI Ambition	22	12.821258	11.695247	Diesel	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda City 2017-2020 EXi	14	11.970350	11.849398	Petrol	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai i20 Sportz Diesel	18	12.323856	11.751942	Diesel	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0
4	Maruti Swift VXI BSIII	15	11.775290	11.695247	Petrol	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0

In [106…

```
lower_bound = 8.517193
upper_bound = 12.128113

# Filter for values within the specified range
df = df[(df['km_driven'] >= lower_bound) & (df['km_driven'] <= upper_bound)]

# Print the filtered DataFrame
df.head()
```

Out[106…

	name	year	selling_price	km_driven	fuel	mileage	engine	max_power	torque	seats
0	Maruti Swift Dzire VDI	22	13.017003	11.887931	Diesel	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda Rapid 1.5 TDI Ambition	22	12.821258	11.695247	Diesel	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda City 2017-2020 EXi	14	11.970350	11.849398	Petrol	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai i20 Sportz Diesel	18	12.323856	11.751942	Diesel	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0
4	Maruti Swift VXI BSIII	15	11.775290	11.695247	Petrol	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0

In [107…

```
df.shape
```

Out[107…

```
(7623, 10)
```

Below I will remove the model names for initial convenience.

In [108…

```
import nltk
```

In [109…

```
def extract_first_word(text):
    words = text.split()
    if words:
        return words[0]
    else:
        return None  # or whatever you want to handle empty rows

# Apply the function to the text column
df['name'] = df['name'].apply(extract_first_word)

df
```

Out[109…

	name	year	selling_price	km_driven	fuel	mileage	engine	max_power	torque	seats
0	Maruti	22	13.017003	11.887931	Diesel	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda	22	12.821258	11.695247	Diesel	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda	14	11.970350	11.849398	Petrol	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai	18	12.323856	11.751942	Diesel	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0
4	Maruti	15	11.775290	11.695247	Petrol	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0
...	...	...	...	...	...	...	...	...	...	...
8123	Hyundai	21	12.676076	11.608236	Petrol	18.5 kmpl	1197 CC	82.85 bhp	113.7Nm@ 4000rpm	5.0
8124	Hyundai	15	11.813030	11.686879	Diesel	16.8 kmpl	1493 CC	110 bhp	24@ 1,900-2,750(kgm@ rpm)	5.0
8125	Maruti	17	12.853176	11.695247	Diesel	19.3 kmpl	1248 CC	73.9 bhp	190Nm@ 2000rpm	5.0
8126	Tata	21	12.577636	10.126631	Diesel	23.57 kmpl	1396 CC	70 bhp	140Nm@ 1800-3000rpm	5.0
8127	Tata	21	12.577636	10.126631	Diesel	23.57 kmpl	1396 CC	70 bhp	140Nm@ 1800-3000rpm	5.0

7623 rows × 10 columns

In [110…

```
import re

def extract_first_word(text):
    if isinstance(text, str):  # Check if the value is a string
        match = re.match(r'^\w+', text)
        return match.group() if match else None
    else:
        return None

df['mileage'] = df['mileage'].apply(extract_first_word)
df['engine'] = df['engine'].apply(extract_first_word)
df['max_power'] = df['max_power'].apply(extract_first_word)

df
```

Out[110...

	name	year	selling_price	km_driven	fuel	mileage	engine	max_power	torque	seats
0	Maruti	22	13.017003	11.887931	Diesel	23	1248	74	190Nm@ 2000rpm	5.0
1	Skoda	22	12.821258	11.695247	Diesel	21	1498	103	250Nm@ 1500-2500rpm	5.0
2	Honda	14	11.970350	11.849398	Petrol	17	1497	78	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai	18	12.323856	11.751942	Diesel	23	1396	90	22.4 kgm at 1750-2750rpm	5.0
4	Maruti	15	11.775290	11.695247	Petrol	16	1298	88	11.5@ 4,500(kgm@ rpm)	5.0
...	...	...	...	...	...	...	...	...	...	...
8123	Hyundai	21	12.676076	11.608236	Petrol	18	1197	82	113.7Nm@ 4000rpm	5.0
8124	Hyundai	15	11.813030	11.686879	Diesel	16	1493	110	24@ 1,900-2,750(kgm@ rpm)	5.0
8125	Maruti	17	12.853176	11.695247	Diesel	19	1248	73	190Nm@ 2000rpm	5.0
8126	Tata	21	12.577636	10.126631	Diesel	23	1396	70	140Nm@ 1800-3000rpm	5.0
8127	Tata	21	12.577636	10.126631	Diesel	23	1396	70	140Nm@ 1800-3000rpm	5.0

7623 rows × 10 columns

In [111...

```
torq = df.torque
```

In [112...

```
df.drop(columns=['torque'], inplace=True)
```

In [113...

```
df.head()
```

Out[113...

	name	year	selling_price	km_driven	fuel	mileage	engine	max_power	seats
0	Maruti	22	13.017003	11.887931	Diesel	23	1248	74	5.0
1	Skoda	22	12.821258	11.695247	Diesel	21	1498	103	5.0
2	Honda	14	11.970350	11.849398	Petrol	17	1497	78	5.0
3	Hyundai	18	12.323856	11.751942	Diesel	23	1396	90	5.0
4	Maruti	15	11.775290	11.695247	Petrol	16	1298	88	5.0

In [114...

```
import pandas as pd
import category_encoders as ce
encoder = ce.OneHotEncoder(cols=['name', 'fuel'], use_cat_names=True)

# Fit and transform the data
df1 = encoder.fit_transform(df)
```

In [115...

```
df1.head(5)
```

Out[115...

	name_Maruti	name_Skoda	name_Honda	name_Hyundai	name_Toyota	name_Ford	name_Renault	name_Mahindra	name_Tata	name_Chevrolet	...
0	1	0	0	0	0	0	0	0	0	0	...
1	0	1	0	0	0	0	0	0	0	0	...
2	0	0	1	0	0	0	0	0	0	0	...
3	0	0	0	1	0	0	0	0	0	0	...
4	1	0	0	0	0	0	0	0	0	0	...

5 rows × 39 columns

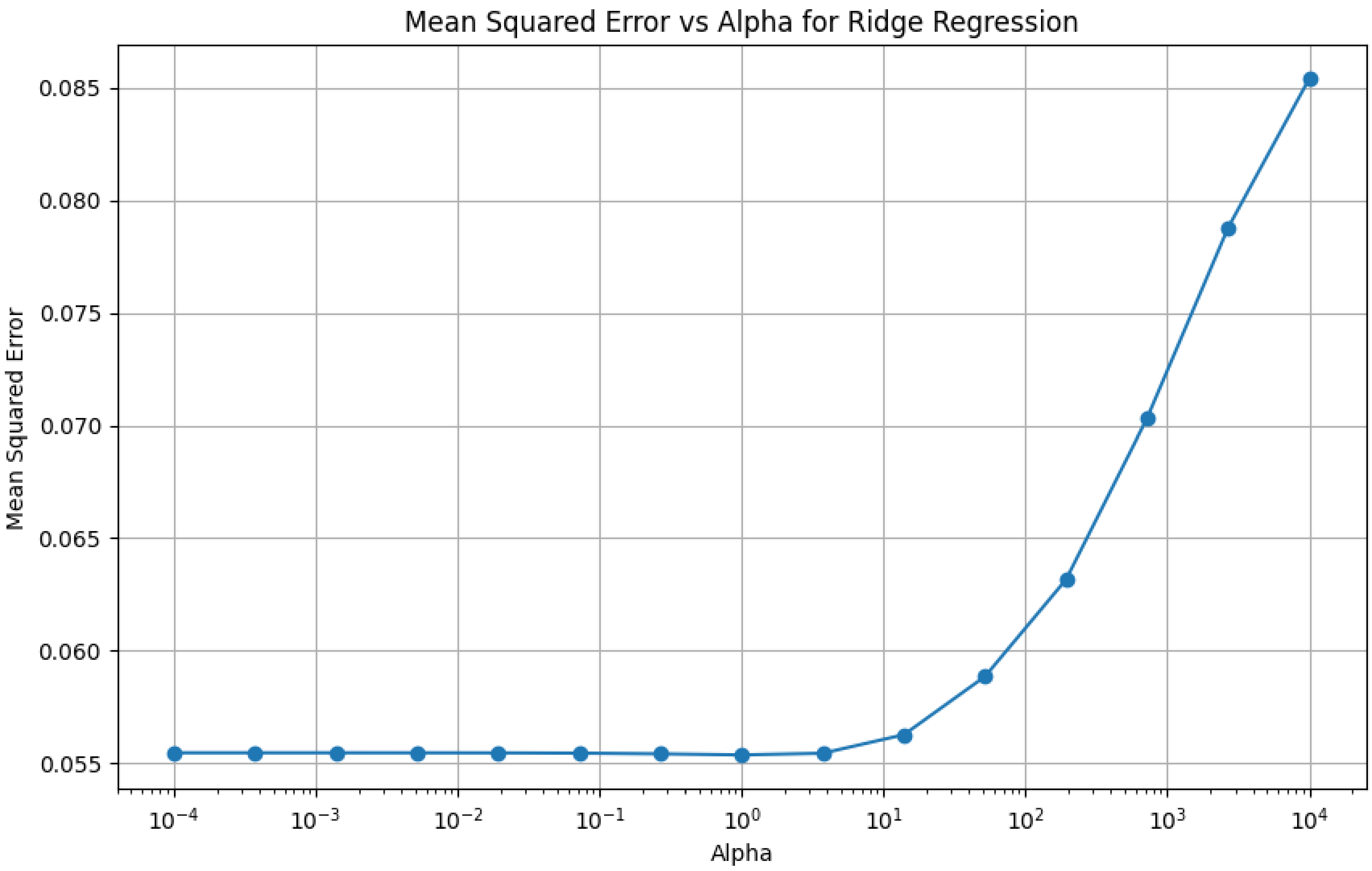
In [116...

```
df1 = df1.dropna()
```

In [117...

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error

# Assuming df1 contains both features and the target variable 'selling_price'
# Extract features (independent variables)
X = df1.drop(columns=['selling_price'])

# Extract target variable
y = df1['selling_price']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define a range of alpha values
alphas = np.logspace(-4, 4, 15)  # Example: 100 alphas from 10^-3 to 10^3

# Initialize lists to store MSE values
mse_values = []
```

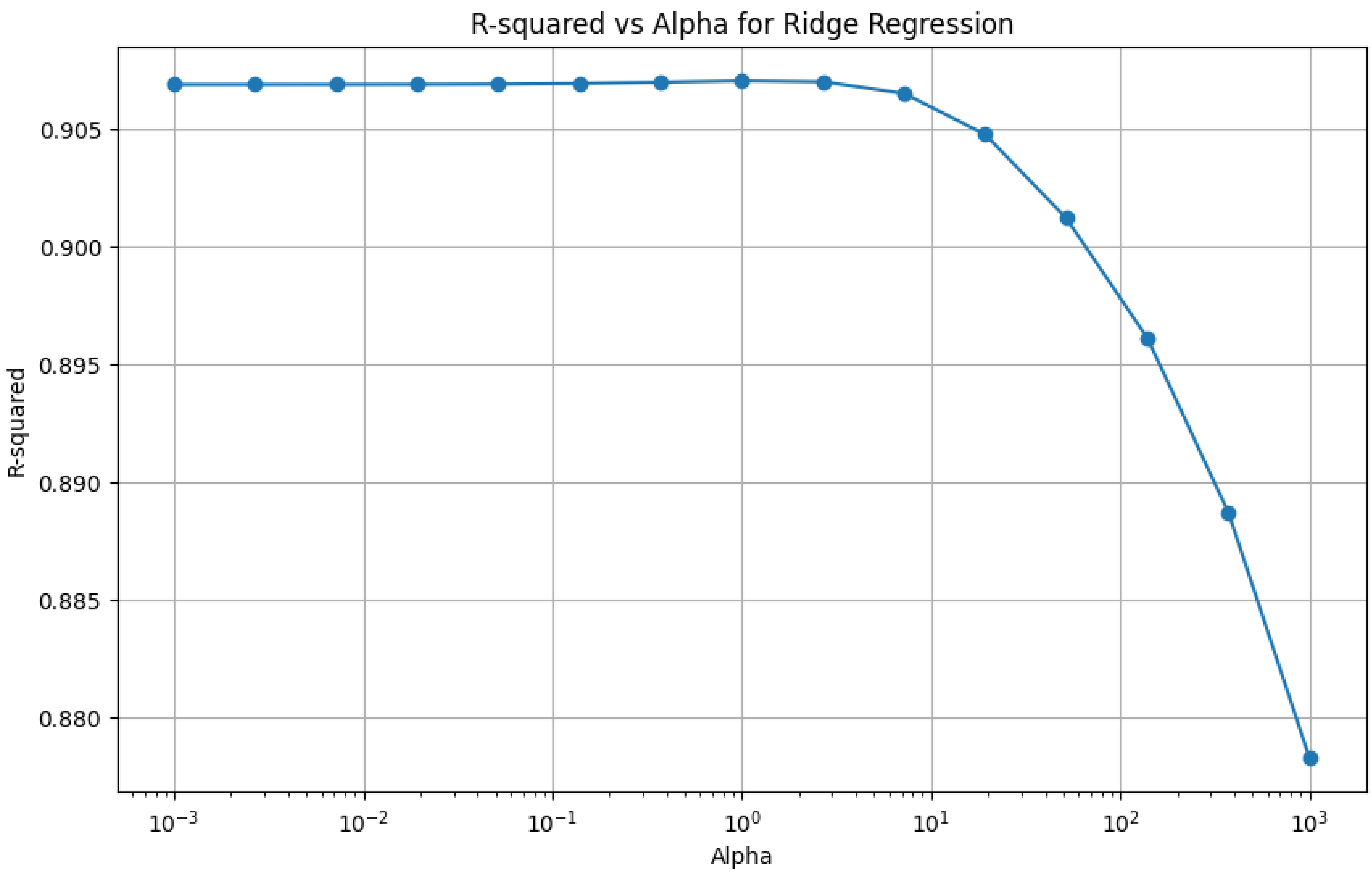
```python
# Iterate over different alpha values
for alpha in alphas:
    # Initialize Ridge regression model with the current alpha
    model = Ridge(alpha=alpha)

    # Train the model on the training data
    model.fit(X_train, y_train)

    # Predict the target variable on the testing data
    y_pred = model.predict(X_test)

    # Calculate Mean Squared Error
    mse = mean_squared_error(y_test, y_pred)

    # Append MSE to the list
    mse_values.append(mse)

# Plot the MSE values for different alpha values
plt.figure(figsize=(10, 6))
plt.plot(alphas, mse_values, marker='o', linestyle='-')
plt.xscale('log')  # Set x-axis to logarithmic scale for better visualization
plt.xlabel('Alpha')
plt.ylabel('Mean Squared Error')
plt.title('Mean Squared Error vs Alpha for Ridge Regression')
plt.grid(True)
plt.show()
```



In [118…

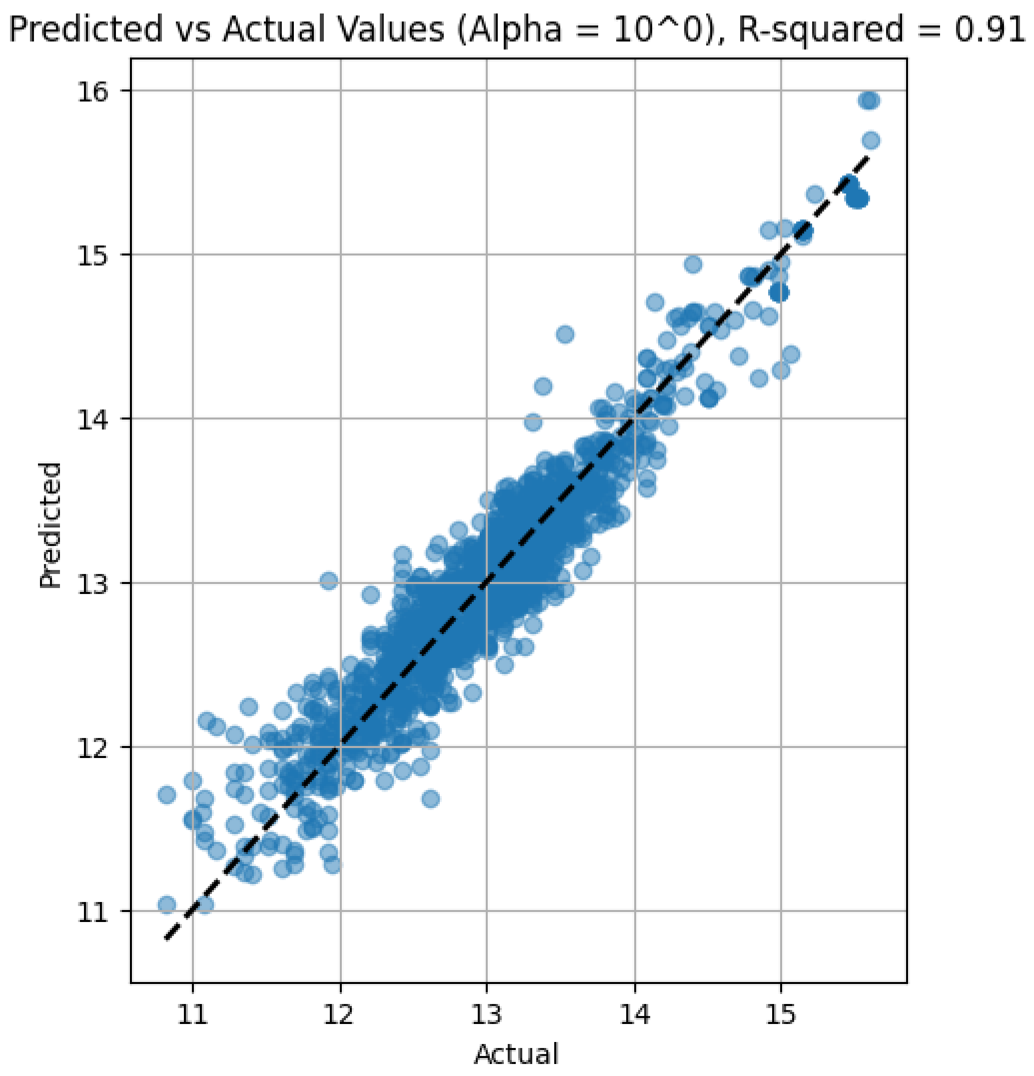
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score

# Assuming df1 contains both features and the target variable 'selling_price'
# Extract features (independent variables)
X = df1.drop(columns=['selling_price'])

# Extract target variable
y = df1['selling_price']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define a range of alpha values
alphas = np.logspace(-3, 3, 15)  # Example: 100 alphas from 10^-3 to 10^3

# Initialize lists to store R-squared values
r2_values = []

# Iterate over different alpha values
for alpha in alphas:
    # Initialize Ridge regression model with the current alpha
    model = Ridge(alpha=alpha)

    # Train the model on the training data
    model.fit(X_train, y_train)

    # Predict the target variable on the testing data
    y_pred = model.predict(X_test)
```

```python
    # Calculate R-squared
    r2 = r2_score(y_test, y_pred)

    # Append R-squared to the list
    r2_values.append(r2)

# Plot the R-squared values for different alpha values
plt.figure(figsize=(10, 6))
plt.plot(alphas, r2_values, marker='o', linestyle='-')
plt.xscale('log')  # Set x-axis to logarithmic scale for better visualization
plt.xlabel('Alpha')
plt.ylabel('R-squared')
plt.title('R-squared vs Alpha for Ridge Regression')
plt.grid(True)
plt.show()
```



In [121…

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score

# Assuming df1 contains both features and the target variable 'selling_price'
# Extract features (independent variables)
X = df1.drop(columns=['selling_price'])

# Extract target variable
y = df1['selling_price']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Set the alpha value
alpha = 1  # equivalent to 10^0

# Initialize Ridge regression model with the specified alpha
model = Ridge(alpha=alpha)

# Train the model on the training data
model.fit(X_train, y_train)

# Predict the target variable on the testing data
y_pred = model.predict(X_test)

# Calculate R-squared
r2 = r2_score(y_test, y_pred)

# Plot predicted vs actual values
plt.figure(figsize=(5, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)  # Add diagonal line
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Predicted vs Actual Values (Alpha = 10^0), R-squared = {:.2f}'.format(r2))
plt.grid(True)
plt.show()
```


Predicted vs Actual Values (Alpha = 10^0), R-squared = 0.91
Predicted
Actual
11
12
13
14
15
16

In [ ]:

In [ ]:

In [ ]:

In [ ]: