Importing Dependencies

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
```

Data Collection Processing

In [2]:
```
df=pd.read_csv(r"C:\Users\USER\Downloads\archive\insurance.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

From above information till now: We now, we do not have null values so no need of Simple Imputer, We have few categorical values so later we may need OneHotEncoder and LabelEncoder, our data - different column have different ranges of values so to bring uniformity we may need minmax scaler or Standardization.

Lets analysed data before converting and changing anything in data

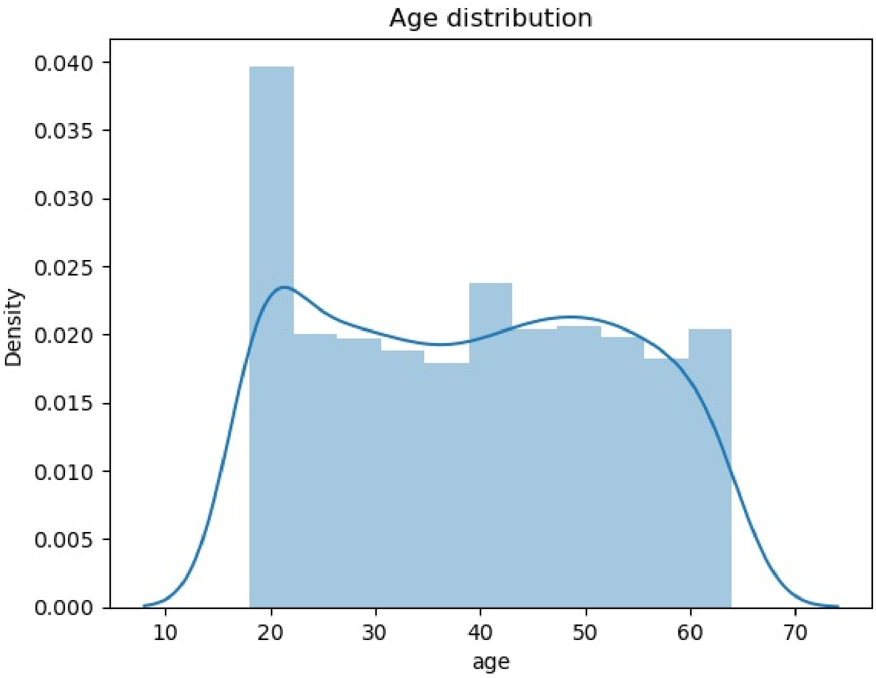
In [6]:
```
df.describe()
```

Out[6]:

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

In [7]:
```
sns.distplot(df['age'])
plt.title('Age distribution')
```

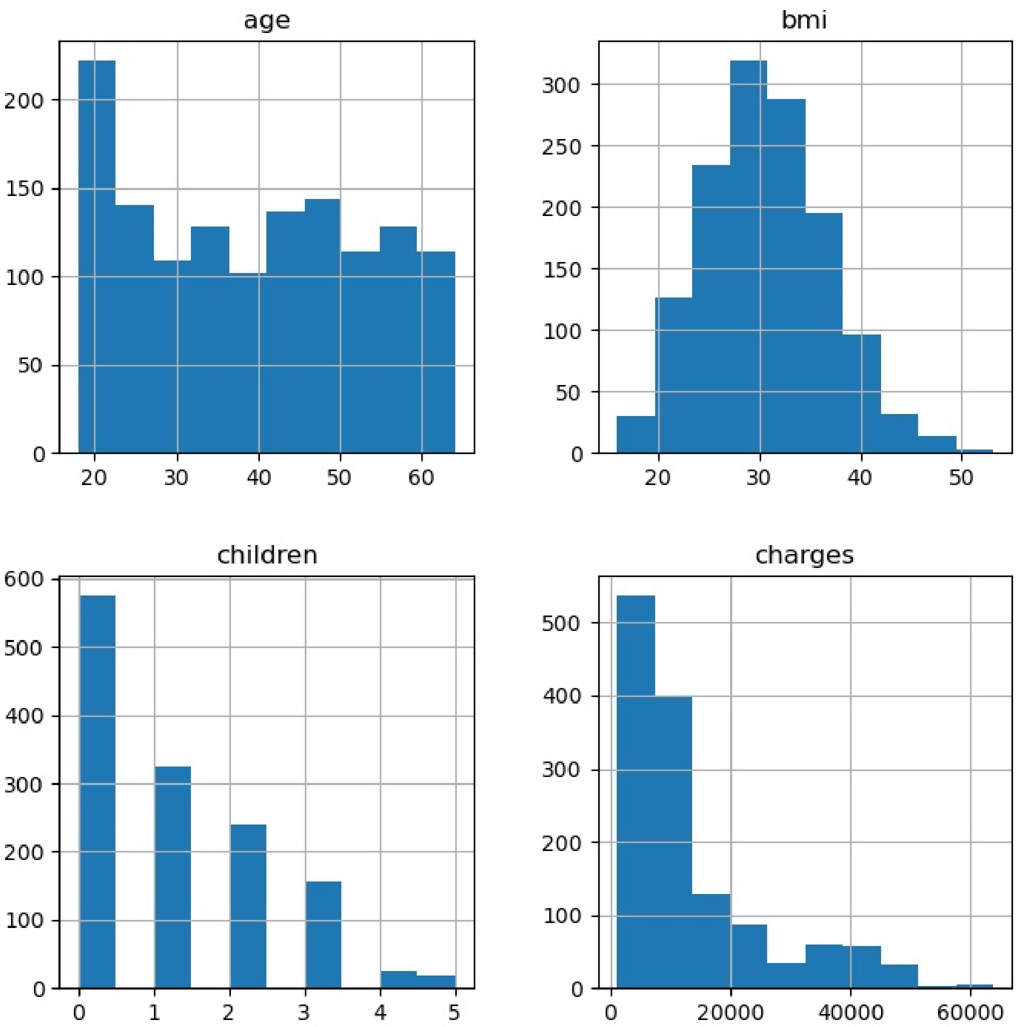
Out[7]:
```
Text(0.5, 1.0, 'Age distribution')
```



We can directly see all plot together too:

In [8]:
```
df.hist(figsize=(8,8))
```

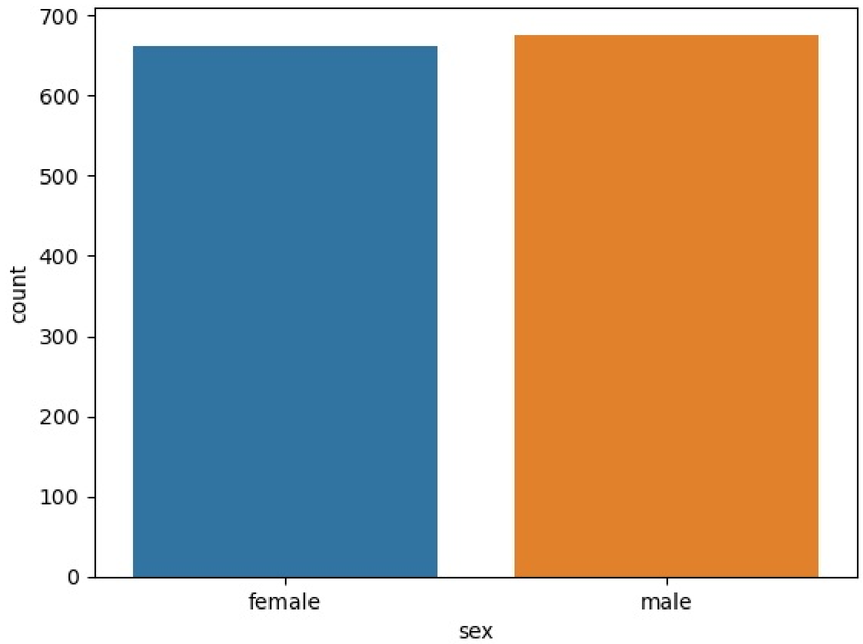
Out[8]:
```
array([[<Axes: title={'center': 'age'}>, <Axes: title={'center': 'bmi'}>],
       [<Axes: title={'center': 'children'}>,
        <Axes: title={'center': 'charges'}>]], dtype=object)
```



For categorical like Gender lets use countplot:

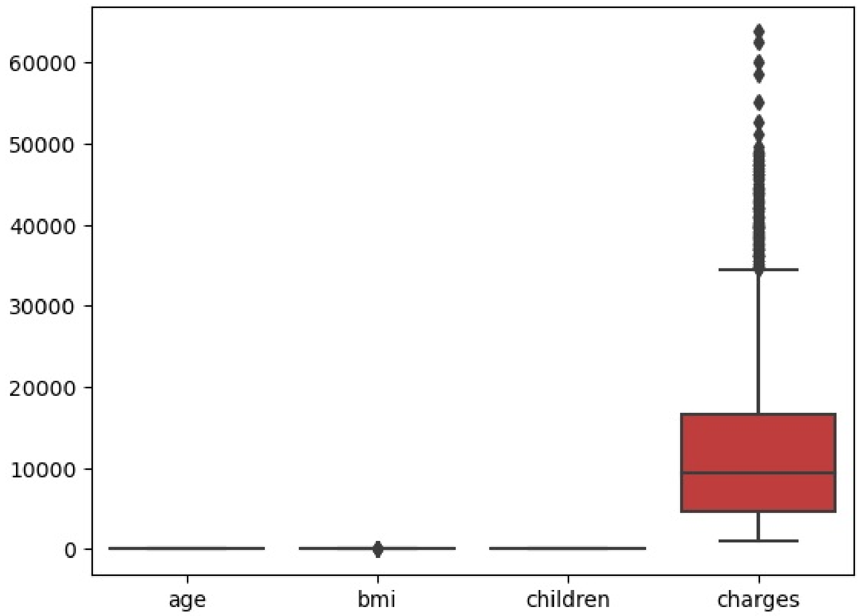
In [9]:
```
sns.countplot(x='sex',data=df)
```

Out[9]:
```
<Axes: xlabel='sex', ylabel='count'>
```



In [10]:
```
sns.boxplot(data=df)
```

Out[10]:
```
<Axes: >
```

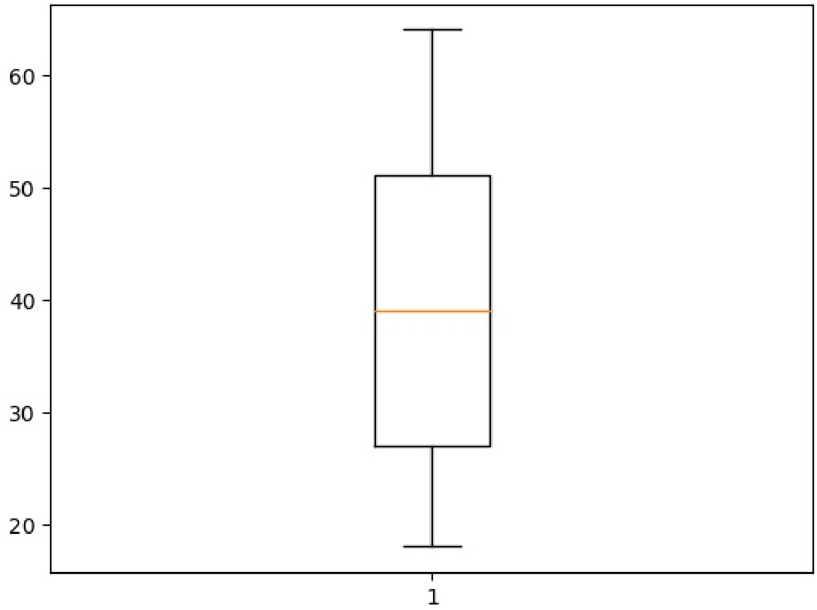


From above boxplot, like if we compare, we can see ranges, like all data range from various column are not uniform.

Lets see individually

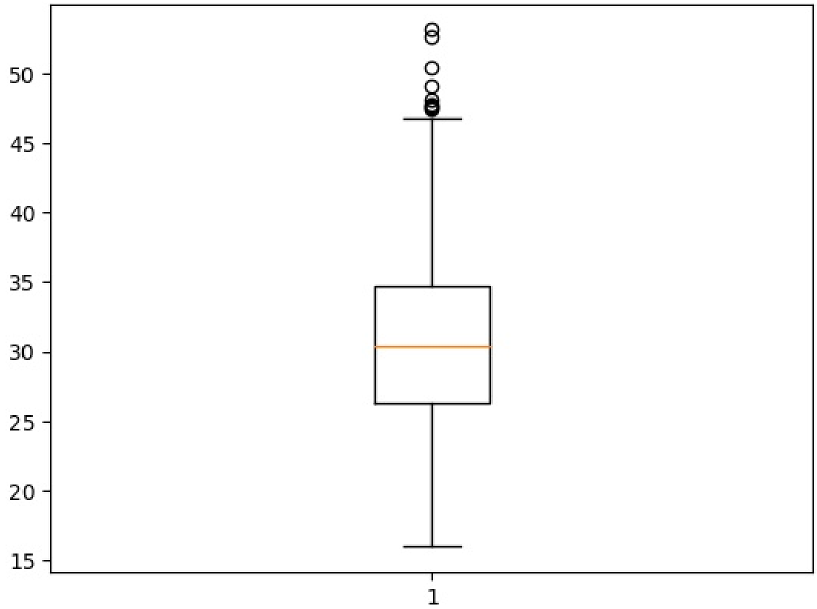
In [11]:
```
plt.boxplot(df['age'])
```

Out[11]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2de0cb23cd0>,
  <matplotlib.lines.Line2D at 0x2de0cb307d0>],
 'caps': [<matplotlib.lines.Line2D at 0x2de0cb31210>,
  <matplotlib.lines.Line2D at 0x2de0cb31d10>],
 'boxes': [<matplotlib.lines.Line2D at 0x2de0cb22f50>],
 'medians': [<matplotlib.lines.Line2D at 0x2de0cb32890>],
 'fliers': [<matplotlib.lines.Line2D at 0x2de0cb33350>],
 'means': []}
```



In [12]:
```
plt.boxplot(df['bmi'])
```

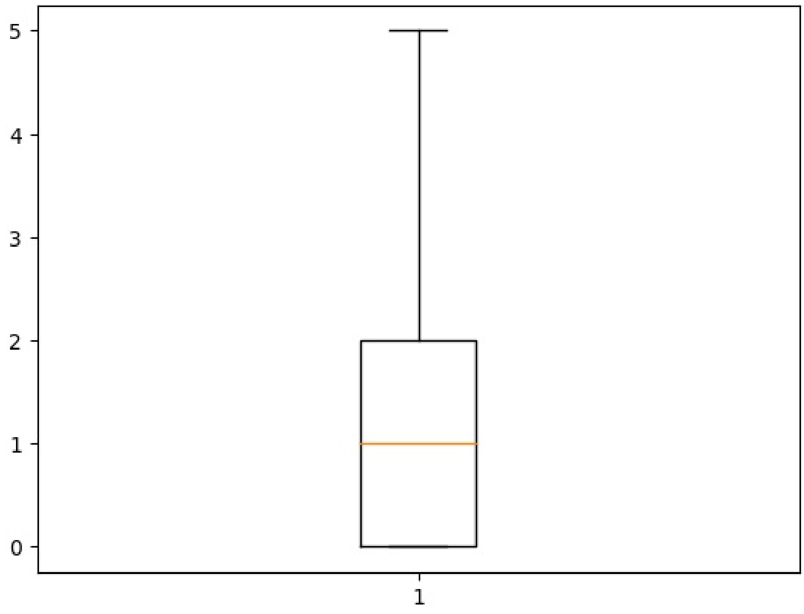
Out[12]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2de0cb8cb10>,
  <matplotlib.lines.Line2D at 0x2de0cb8d690>],
 'caps': [<matplotlib.lines.Line2D at 0x2de0cb8e2d0>,
  <matplotlib.lines.Line2D at 0x2de0cb8ec50>],
 'boxes': [<matplotlib.lines.Line2D at 0x2de0cb83d90>],
 'medians': [<matplotlib.lines.Line2D at 0x2de0cb8f6d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x2de0cb5f390>],
 'means': []}
```



In bmi, we can see some outlier.

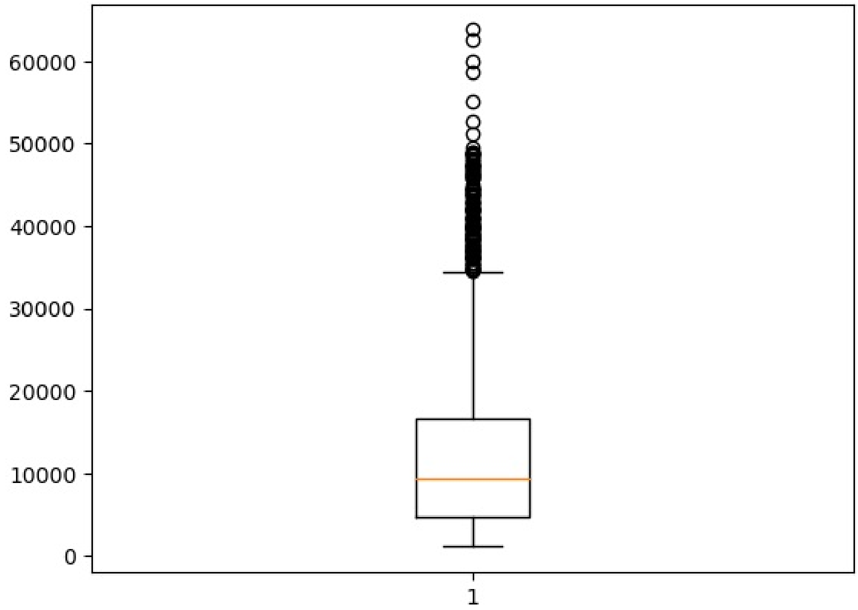
In [13]:
```
plt.boxplot(df['children'])
```

Out[13]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2de0c9b7b50>,
  <matplotlib.lines.Line2D at 0x2de0c9c8610>],
 'caps': [<matplotlib.lines.Line2D at 0x2de0c9c9150>,
  <matplotlib.lines.Line2D at 0x2de0c9c9c10>],
 'boxes': [<matplotlib.lines.Line2D at 0x2de0c9b6e90>],
 'medians': [<matplotlib.lines.Line2D at 0x2de0c9ca6d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x2de0cbeb8d0>],
 'means': []}
```



In [14]:
```python
plt.boxplot(df['charges'])
```

Out[14]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2de0ca1ead0>,
  <matplotlib.lines.Line2D at 0x2de0ca1f710>],
 'caps': [<matplotlib.lines.Line2D at 0x2de0ca28390>,
  <matplotlib.lines.Line2D at 0x2de0ca28e90>],
 'boxes': [<matplotlib.lines.Line2D at 0x2de0c9b6850>],
 'medians': [<matplotlib.lines.Line2D at 0x2de0ca29910>],
 'fliers': [<matplotlib.lines.Line2D at 0x2de0ca2a3d0>],
 'means': []}
```



Model Building

In [15]:
```python
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler, OneHotEncoder, StandardScaler, MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder
```

In [16]:
```python
# Columns to encode
categorical_columns = ['sex', 'smoker', 'region']
```

In [17]:
```python
# Apply LabelEncoder for binary categorical columns
df['sex'] = LabelEncoder().fit_transform(df['sex'])
df['smoker'] = LabelEncoder().fit_transform(df['smoker'])
```

In [18]:
```python
# Apply OneHotEncoder for multi-category columns
```

```
df = pd.get_dummies(df, columns=['region'], drop_first=True)
```

Lets see how our data looks now

In [19]:
```
df.head()
```

Out[19]:

	age	sex	bmi	children	smoker	charges	region_northwest	region_southeast	region_southwest
0	19	0	27.900	0	1	16884.92400	0	0	1
1	18	1	33.770	1	0	1725.55230	0	1	0
2	28	1	33.000	3	0	4449.46200	0	1	0
3	33	1	22.705	0	0	21984.47061	1	0	0
4	32	1	28.880	0	0	3866.85520	1	0	0

In [20]:
```
# Separate features and target variable
X = df.drop(columns=['charges'])
y = df['charges']
```

We separated it because now we will do scaling and we cannot scale target variable.

In [21]:
```
# Define ColumnTransformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', RobustScaler(), ['age', 'bmi', 'children']),
        ('cat', 'passthrough', ['sex', 'smoker', 'region_northwest', 'region_southeast', 'region_southwest'])
    ]
)
```

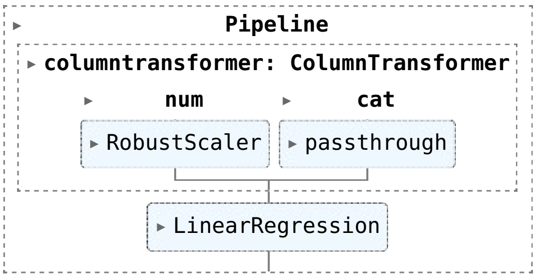
Model Building using Pipeline

In [22]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [23]:
```
# Create a pipeline
pipeline = make_pipeline(preprocessor, LinearRegression())
```

In [24]:
```
# Fit the model
pipeline.fit(X_train, y_train)
```

Out[24]:



In [25]:
```
# Make predictions
y_pred = pipeline.predict(X_test)
```

In [26]:
```
print(y_pred)
```

```
[ 8969.55027444  7068.74744287 36858.41091155  9454.67850053
 26973.17345656 10864.11316424   170.28084136 16903.45028662
  1092.43093614 11218.34318352 28101.68455267  9377.73460205
  5263.0595179  38416.04221107 40255.82339284 37098.25353123
 15240.39392306 35912.88264434  9112.52398703 31461.92108909
  3847.68845883 10130.12001517  2370.54189389  7140.21550828
 11301.76782638 12961.65366224 14509.47251876  6159.8976107
  9963.85857263  2177.85718217  9115.93673493 13073.68932159
  4561.82376202  3408.20756033  4459.81359745 13032.06505076
  1979.99357292  8813.28303302 33271.29124448 32585.51583927
  3908.76090964  4326.10774721 14142.81326533 11423.45494846
  8774.13955311 12097.28051001  5281.57353499  3150.5596042
 35494.46461214  9150.1124786  15836.84575621  2343.57470069
 12364.78414194  1482.29488266 13389.06105161 12573.57395972
  4341.83680558 32165.33688042 13321.3360032  12896.82071102
 14167.99421483 10506.17623512 16360.78543548  7763.89824584
 11839.25019431  4061.19750503 26652.40230125 10930.14138671
  2137.41385988  6209.01123411 10729.82391284 11628.3104129
 10981.04528946  9166.50818596 11954.27732874  6747.85121734
  7248.5304713  10735.16710748  6580.84819774  8762.00329355
  3767.13383454 36632.4975496   6378.11979721 30842.09248656
 34846.52451051 35278.07387112  7019.444352   12861.38414264
  9942.30149778 14473.5260648  17693.37304474 35258.24845137
 33029.58968269  6185.91730447 31999.98962535  9481.33158273
 29444.04271523  3674.48498404 28308.26432106  5823.36495229
  5407.76752001  1883.4947576  11499.675042   15075.90690632
 11699.63163008  4308.82427855  9895.1840044  31708.40056201
   -86.87094667 32819.71429004  3280.69178415 10183.88853878
 14318.76389179 31642.35684542 11461.57806791  3929.23701831
 13107.89313088 31810.99450607  8152.02593593  3238.08417076
  8439.56108376 10594.63871458 15219.68736374  5647.8808143
  3781.95285499 10228.944897   10900.12933883 11122.74845192
 14438.14112575  7430.31504776  5386.22676759  9231.32739901
  9343.76283713 12538.27606344  8337.66982683 15333.36900871
  8411.2145439  31797.27496298 35785.91843418 31603.71967017
  6011.96229251 12607.03584641  6013.5115031  14560.79590559
  2493.47989441 32963.45524228  6265.14380504  5034.62173797
 14344.81347407  6941.1412259  38670.01270366  3087.58741836
  5885.8752536  31686.24200595 11562.61859836  8476.04749512
 14806.72486264  9814.46186143 27105.71831469 33453.83352069
 14551.8999207   1684.36856768 13166.96197398  2222.76894041
  5449.59393727 11568.96325488 39807.96912709 36500.65163031
 34001.37945748  3897.27856532  7456.14132125  8661.82084477
 12450.92458882  4813.53293089  2047.65528159 32112.11251984
 25111.52085938 17484.27663755 26411.46181822 10159.52421
 37260.32666386  -441.23918333  6779.55013103  7781.45337795
  4367.95988484  5105.87170813  5919.18675042  4305.71645941
 15191.08806502 11132.09935114  6932.80116584  2525.64793222
  1536.05183213 31944.78284317 16414.12251517 12011.53367195
  1268.05926603 12531.25953189  1564.93415917  8737.33621694
  1873.03940488 33916.22971211 10858.38635063  2603.43633853
 25674.40250332 26343.43022704  9430.91152033  1800.73500777
 13261.42480211  1120.17810533 10386.66427709 10567.29006474
 16944.25995713 26846.54662457  6939.11178393  5193.04710054
  5846.00017265 13229.60536846 11098.33930228  8362.28134289
  5135.53940151 12308.34064139 13861.17886997 35773.70926219
  4157.01930317 28917.86562624  -914.37342357  2873.71150671
 11046.2540774  15683.06950225  5210.67532324  6888.38518351
  3854.31140958 31312.64705453  7241.43226665 12405.99508651
  5619.17039188  9528.22557021 36314.009043    4429.40596906
  9667.91523953 31161.15738995  5747.13292318  4603.37294255
  1048.35533791  4832.6604097   4574.9041044   6507.30666036
 18659.12407756 -1545.57184934  2376.4352498  10694.62157146
  3151.28919904 10209.96361187  3733.89128353  5125.08103172
 12400.90700504  6218.65296628  8231.63765089  7590.50155269
  8924.15352268 10482.90359975 27808.04576398 39061.50093248
 11761.4991981   7687.56363151 40920.29151165 12318.58665305]
```

Evaluate the model

In [27]:
```python
# Evaluate the model
print("Mean Absolute Error:", metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R^2 Score:", metrics.r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 4181.194473753652
Mean Squared Error: 33596915.85136148
Root Mean Squared Error: 5796.284659276275
R^2 Score: 0.7835929767120722
```

Lets more refine the model

In [28]:
```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
```

In [29]:
```python
# Create a pipeline with RandomForest
pipeline_rf = make_pipeline(preprocessor, RandomForestRegressor(n_estimators=100, random_state=42))
```

In [30]:
```
# Evaluate using cross-validation
scores = cross_val_score(pipeline_rf, X, y, cv=5, scoring='r2')
```

In [31]:
```
# Print the average R^2 score
print("Average R^2 Score:", np.mean(scores))
```

```
Average R^2 Score: 0.8366066075842407
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js