```
In [54]: #!pip install kagglehub
```

```
In [1]: import kagglehub

        # Download latest version
        path = kagglehub.dataset_download("camnugent/california-housing-prices")

        print("Path to dataset files:", path)
```

```
Path to dataset files: C:\Users\MOHAMMED HAYATH RK\.cache\kagglehub\datasets\ca
mnugent\california-housing-prices\versions\1
```

```
In [2]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns

        from warnings import filterwarnings
        filterwarnings('ignore')
```

```
In [3]: import os
```

```
In [4]: # List files
        files = os.listdir(path)
        print(files)
```

```
['housing.csv']
```

```
In [5]: path
```

```
Out[5]: 'C:\\Users\\MOHAMMED HAYATH RK\\.cache\\kagglehub\\datasets\\camnugent\\calif
        ornia-housing-prices\\versions\\1'
```

```
In [6]: # Use the path returned by kagglehub
        base_path = path  # path from kagglehub
```

```
In [7]: housing = pd.read_csv(os.path.join(base_path, "housing.csv"))
        housing.head()
```

Out[7]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	popul
0	-122.23	37.88	41.0	880.0	129.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2
2	-122.24	37.85	52.0	1467.0	190.0	
3	-122.25	37.85	52.0	1274.0	235.0	
4	-122.25	37.85	52.0	1627.0	280.0	

Basic EDA

In [8]:
```
housing.shape
```

Out[8]:
```
(20640, 10)
```

In [9]:
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [10]:
```
housing.describe()
```

Out[10]:

	longitude	latitude	housing_median_age	total_rooms	total_bec
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.
mean	-119.569704	35.631861	28.639486	2635.763081	537.
std	2.003532	2.135952	12.585558	2181.615252	421.
min	-124.350000	32.540000	1.000000	2.000000	1.
25%	-121.800000	33.930000	18.000000	1447.750000	296.
50%	-118.490000	34.260000	29.000000	2127.000000	435.
75%	-118.010000	37.710000	37.000000	3148.000000	647.
max	-114.310000	41.950000	52.000000	39320.000000	6445.

In [11]:
```
housing.isnull().sum()
```

Out[11]:
```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

Best way to handle missing values is to use Median Imputation

Why? Because

- Median works better when the data has outliers
- Avoids the shifting the distribution
- Keeps model performance stable

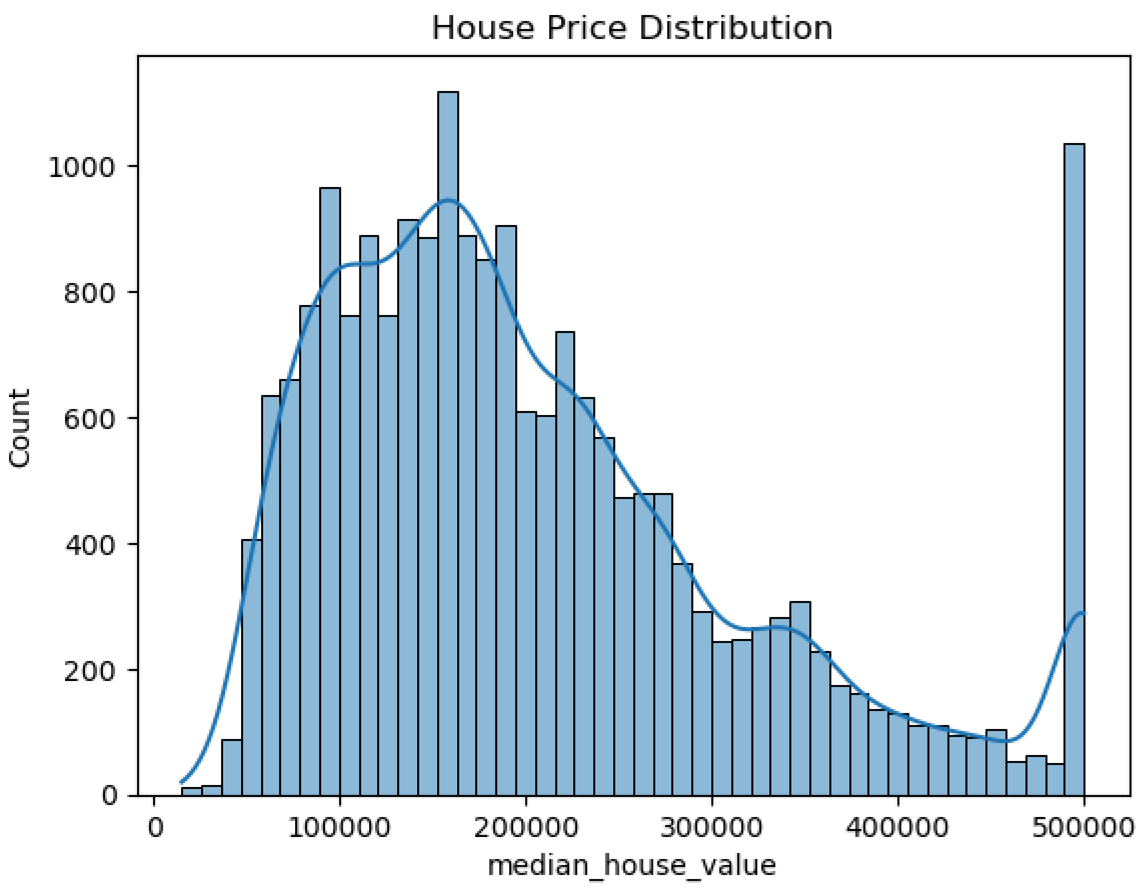
In [12]:
```
# Fill missing values in total_bedrooms using median
housing['total_bedrooms'] = housing['total_bedrooms'].fillna(housing['total_be
```

In [13]:
```
housing.isnull().sum()
```

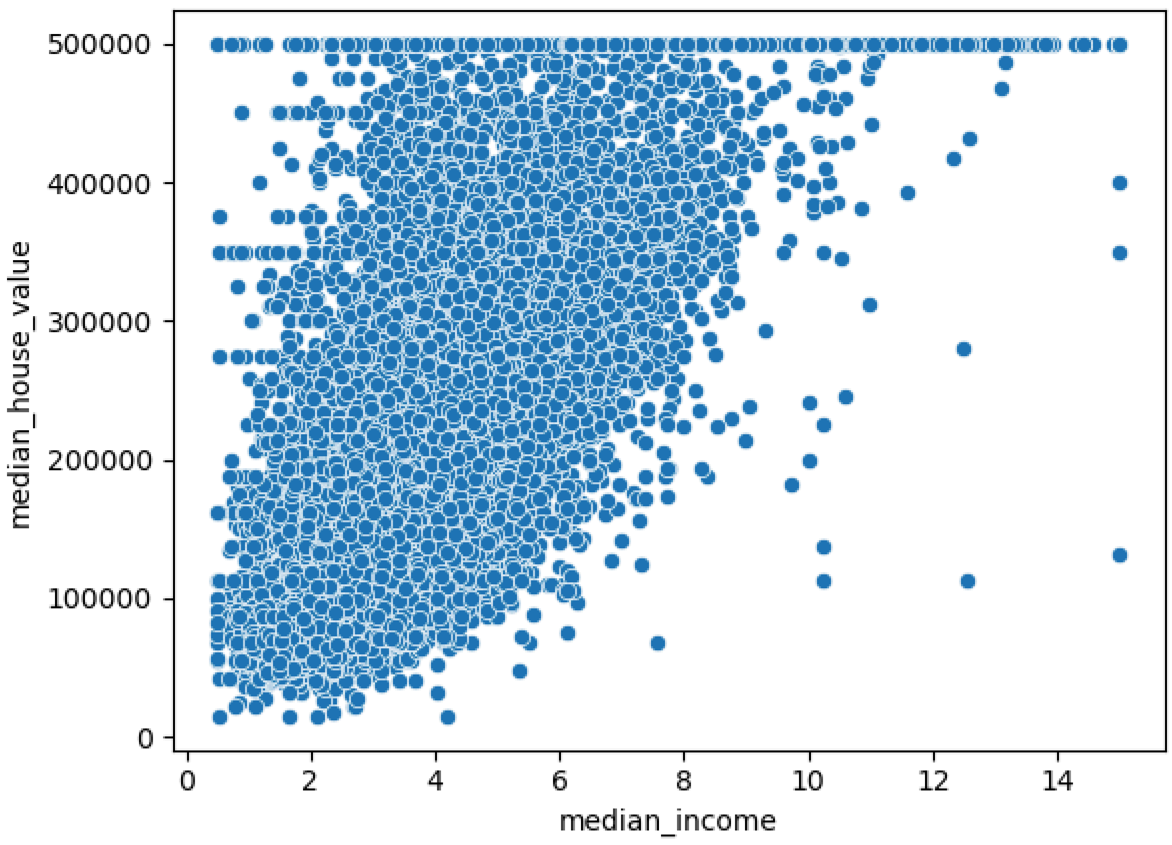
Out[13]:
```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

1. Distribution of target variable

In [14]:
```
sns.histplot(housing["median_house_value"], kde=True)
plt.title("House Price Distribution")
plt.show()
```



In [15]:
```
# using Sactter Plot
sns.scatterplot(data=housing, x="median_income", y="median_house_value")
plt.show()
```



Feature Engineering

a. Define X and y

In [16]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [17]:
```python
X = housing.drop('median_house_value', axis = 1)
y = housing['median_house_value']
```

In [18]:
```python
X
```

Out[18]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
0	-122.23	37.88	41.0	880.0	129.0
1	-122.22	37.86	21.0	7099.0	1106.0
2	-122.24	37.85	52.0	1467.0	190.0
3	-122.25	37.85	52.0	1274.0	235.0
4	-122.25	37.85	52.0	1627.0	280.0
...	...	...	...	...	...
20635	-121.09	39.48	25.0	1665.0	374.0
20636	-121.21	39.49	18.0	697.0	150.0
20637	-121.22	39.43	17.0	2254.0	485.0
20638	-121.32	39.43	18.0	1860.0	409.0
20639	-121.24	39.37	16.0	2785.0	616.0

20640 rows × 9 columns

In [19]:
```
y
```

Out[19]:
```
0        452600.0
1        358500.0
2        352100.0
3        341300.0
4        342200.0
           ...
20635     78100.0
20636     77100.0
20637     92300.0
20638     84700.0
20639     89400.0
Name: median_house_value, Length: 20640, dtype: float64
```

b. Column Sepration

In [20]:
```
numeric_cols = X.select_dtypes(include=['int64', 'float64']).columns
cat_cols = ["ocean_proximity"]
```

In [21]:
```
numeric_cols
```

Out[21]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income'],
      dtype='object')
```

In [22]:
```
cat_cols
```

Out[22]:
```
['ocean_proximity']
```

c. Preprocessing PipeLine

In [23]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_cols),
        ("cat", OneHotEncoder(handle_unknown='ignore'), cat_cols)
    ]
)
```

Traing and Testing Test

In [24]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rar
```

In [25]:
```
X_train
```

Out[25]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
14196	-117.03	32.71	33.0	3126.0	627.0
8267	-118.16	33.77	49.0	3382.0	787.0
17445	-120.48	34.66	4.0	1897.0	331.0
14265	-117.11	32.69	36.0	1421.0	367.0
2271	-119.80	36.78	43.0	2382.0	431.0
...	...	...	...	...	...
11284	-117.96	33.78	35.0	1330.0	201.0
11964	-117.43	34.02	33.0	3084.0	570.0
5390	-118.38	34.03	36.0	2101.0	569.0
860	-121.96	37.58	15.0	3575.0	597.0
15795	-122.42	37.77	52.0	4226.0	1315.0

16512 rows × 9 columns

In [26]:
```
X_test
```

Out[26]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
20046	-119.01	36.06	25.0	1505.0	435.0
3024	-119.46	35.14	30.0	2943.0	435.0
15663	-122.44	37.80	52.0	3830.0	435.0
20484	-118.72	34.28	17.0	3051.0	435.0
9814	-121.93	36.62	34.0	2351.0	435.0
...	...	...	...	...	...
15362	-117.22	33.36	16.0	3165.0	482.0
16623	-120.83	35.36	28.0	4323.0	886.0
18086	-122.05	37.31	25.0	4111.0	538.0
2144	-119.76	36.77	36.0	2507.0	466.0
3665	-118.37	34.22	17.0	1787.0	463.0

4128 rows × 9 columns

In [27]:
```
y_train
```

Out[27]:
```
14196    103000.0
8267     382100.0
17445    172600.0
14265     93400.0
2271      96500.0
           ...
11284    229200.0
11964     97800.0
5390     222100.0
860      283500.0
15795    325000.0
Name: median_house_value, Length: 16512, dtype: float64
```

In [28]:
```
y_test
```

Out[28]:
```
20046     47700.0
3024      45800.0
15663    500001.0
20484    218600.0
9814     278000.0
           ...
15362    263300.0
16623    266800.0
18086    500001.0
2144      72300.0
3665     151500.0
Name: median_house_value, Length: 4128, dtype: float64
```

Models Training

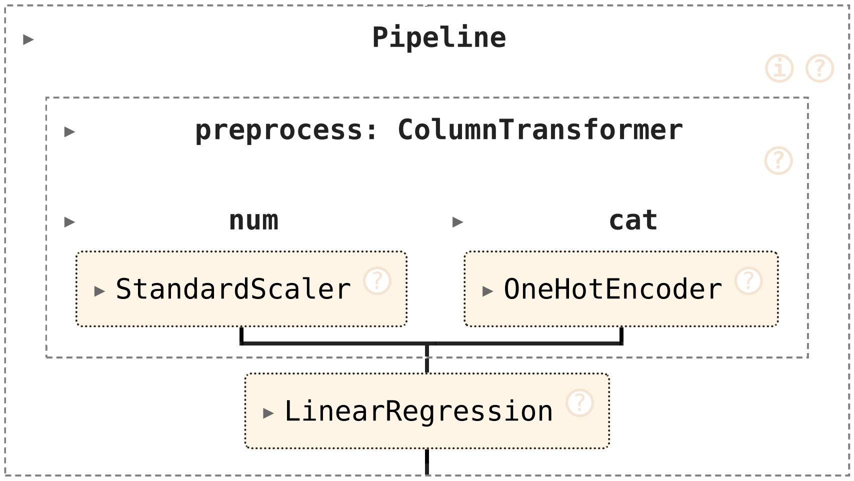
```
In [ ]: model = {}
```

1. Linear reggression

```
In [29]: lr_model = Pipeline(steps = [
             ('preprocess', preprocessor),
             ('model', LinearRegression())
         ])
```

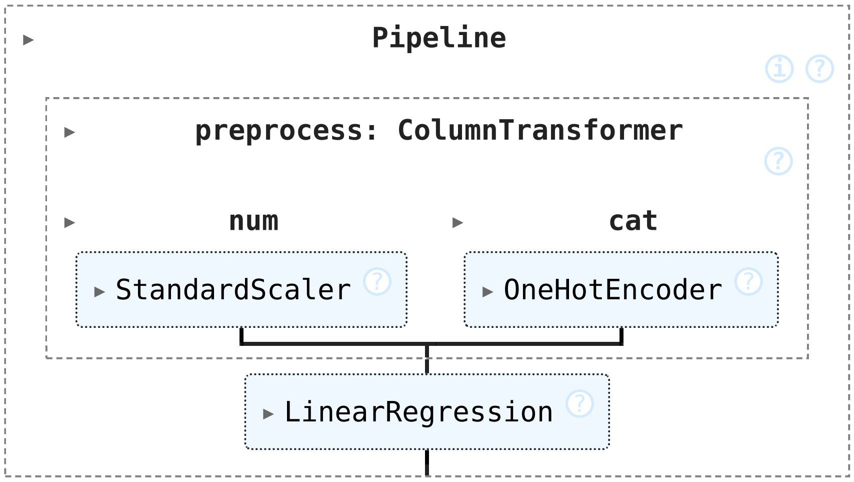
```
In [30]: lr_model
```

Out[30]:



```
In [45]: lr_model.fit(X_train, y_train)
```

Out[45]:



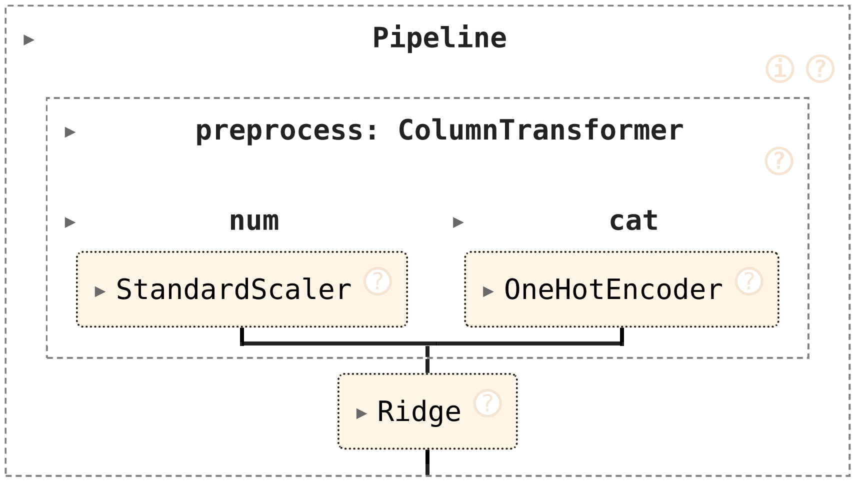
```
In [46]: models["Linear Regression"] = lr_model
```

2. Ridge Regression

In [31]:
```
ridge_model = Pipeline(steps=[
    ('preprocess', preprocessor),
    ('model', Ridge(alpha = 1.0))
])
```

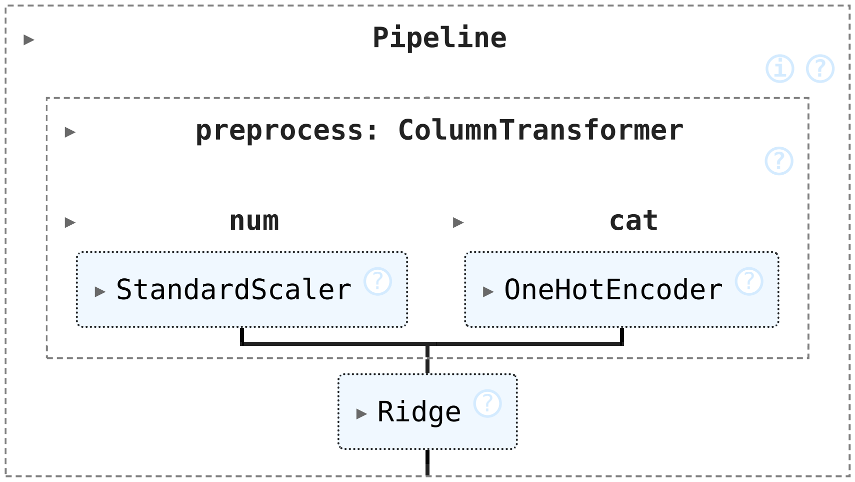
In [32]:
```
ridge_model
```

Out[32]:



In [47]:
```
ridge_model.fit(X_train, y_train)
```

Out[47]:



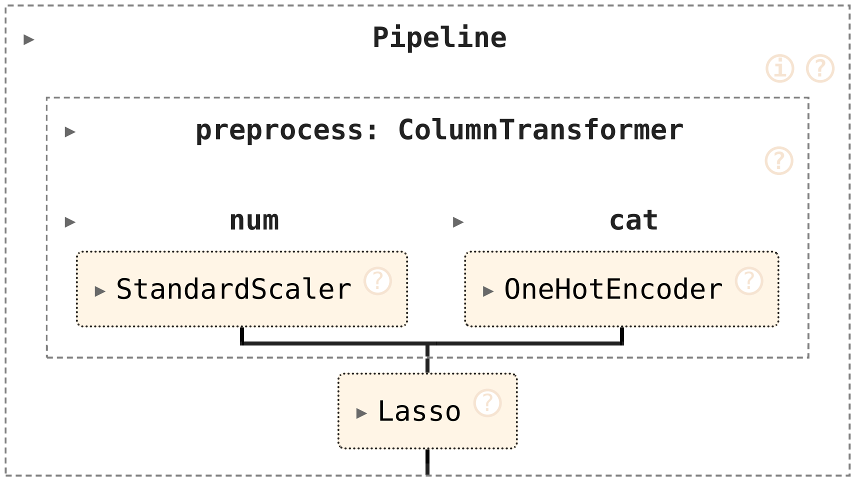
In [48]:
```
models["Ridge Regression"] = ridge_model
```

3. Lasso Regression

In [33]:
```
lasso_model = Pipeline(steps=[
    ('preprocess', preprocessor),
    ('model', Lasso(alpha=0.0001))
])
```

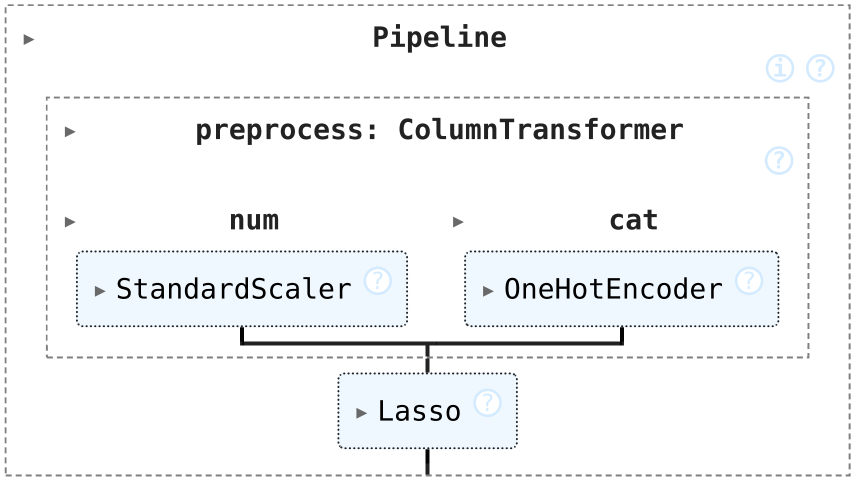
In [34]:
```
lasso_model
```

Out[34]:



In [35]:
```
lasso_model.fit(X_train, y_train)
```

Out[35]:



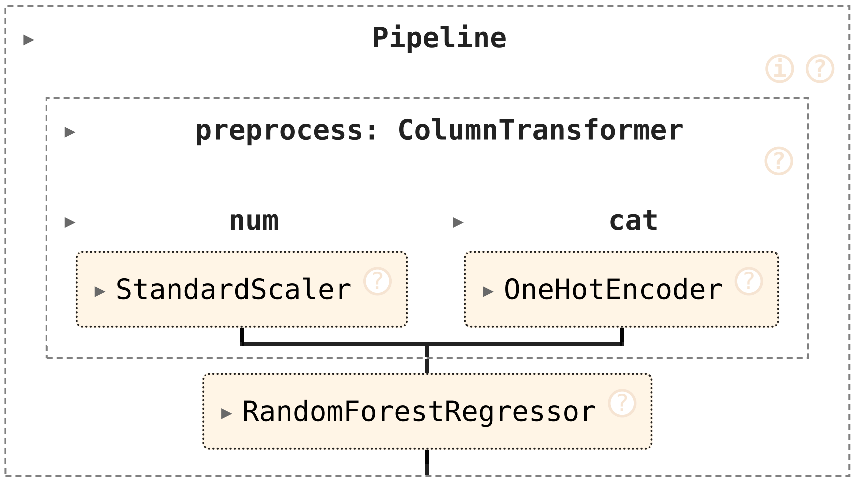
In [49]:
```
models["Lasso Regression"] = lasso_model
```

4. Random Forest Regressor

In [36]:
```
rf_model = Pipeline(steps=[
    ('preprocess', preprocessor),
    ('model', RandomForestRegressor(n_estimators=200, random_state=42))
])
```

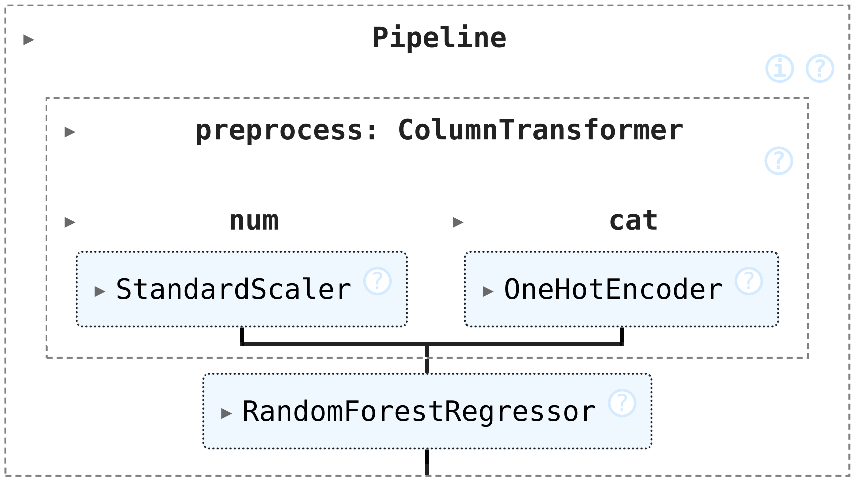
In [37]:
```
rf_model
```

Out[37]:



In [38]:
```
rf_model.fit(X_train, y_train)
```

Out[38]:



In [50]:
```
models["Random Forest"] = rf_model
```

MODEL EVALUATION FUNCTION

In [39]:
```
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    return mae, mse, rmse, r2
```

CHECKING MODEL SCORES

In [51]:
```
# models = {
```

```
#    "Linear Regression": lr_model,
#    "Ridge Regression": ridge_model,
#    "Lasso Regression": lasso_model,
#    "Random Forest": rf_model
#}

results = {}

for name, model in models.items():
    mae, mse, rmse, r2 = evaluate_model(model, X_test, y_test)
    results[name] = [mae, mse, rmse, r2]

results_df = pd.DataFrame(results, index=["MAE", "MSE", "RMSE", "R2 Score"])
results_df
```

Out[51]:

	Linear Regression	Ridge Regression	Lasso Regression	Random Forest
MAE	5.067074e+04	5.067717e+04	5.067074e+04	3.146493e+04
MSE	4.908477e+09	4.909433e+09	4.908477e+09	2.379640e+09
RMSE	7.006052e+04	7.006735e+04	7.006052e+04	4.878156e+04
R2 Score	6.254241e-01	6.253511e-01	6.254241e-01	8.184047e-01

Conclusion

In [52]:

```
print("Best Model Based on R2 Score:")
print(results_df.loc["R2 Score"].idxmax())
```

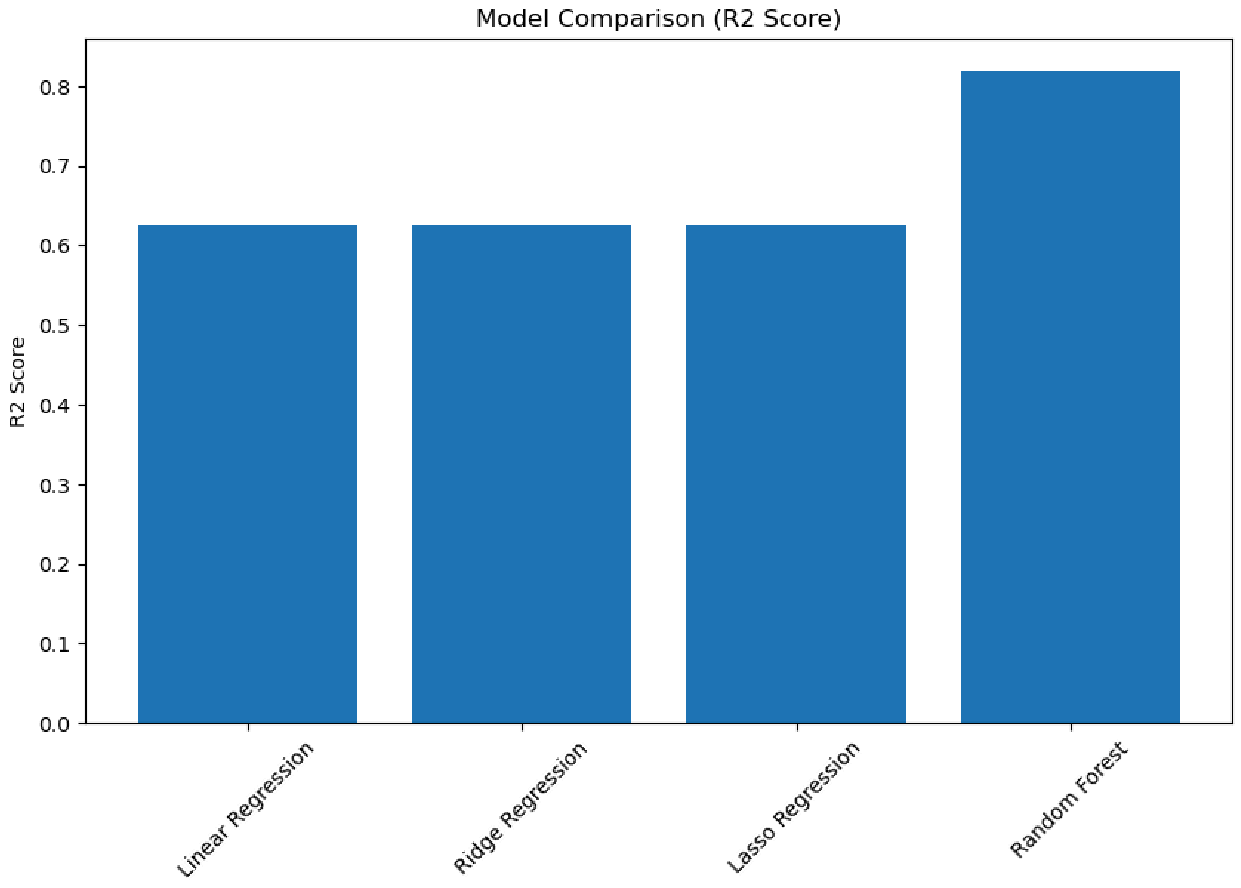
```
Best Model Based on R2 Score:
Random Forest
```

Model Comparison Chart

In [55]:

```
# Extract R2 scores from results table
r2_scores = results_df.loc["R2 Score"]

plt.figure(figsize=(10,6))
plt.bar(r2_scores.index, r2_scores.values)
plt.ylabel("R2 Score")
plt.title("Model Comparison (R2 Score)")
plt.xticks(rotation=45)
plt.show()
```


Model Comparison (R2 Score)
R2 Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Linear Regression
Ridge Regression
Lasso Regression
Random Forest