In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries i
# It is defined by the kaggle/python Docker image: https://github.com/kag
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) wil

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) th
# You can also write temporary files to /kaggle/temp/, but they won't be
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_s
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import RandomizedSearchCV
```

In [3]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [14]:
```
dataset = pd.read_csv("housing.csv")
```

# DATA EXPLORATION

In [15]:
```
dataset.head()
```

Out[15]:

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | hc |
|---|---|---|---|---|---|---|---|
| **0** | -122.23 | 37.88 | 41.0 | 880.0 | 129.0 | 322.0 | |
| **1** | -122.22 | 37.86 | 21.0 | 7099.0 | 1106.0 | 2401.0 | |
| **2** | -122.24 | 37.85 | 52.0 | 1467.0 | 190.0 | 496.0 | |
| **3** | -122.25 | 37.85 | 52.0 | 1274.0 | 235.0 | 558.0 | |
| **4** | -122.25 | 37.85 | 52.0 | 1627.0 | 280.0 | 565.0 | |

In [16]:
```
print(dataset.shape)
```

```
(20640, 10)
```

In [17]:
```
dataset.describe()
```

Out[17]:

| | longitude | latitude | housing_median_age | total_rooms | total_bedroom |
|---|---|---|---|---|---|
| **count** | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20433.00000 |
| **mean** | -119.569704 | 35.631861 | 28.639486 | 2635.763081 | 537.87055 |
| **std** | 2.003532 | 2.135952 | 12.585558 | 2181.615252 | 421.38507 |
| **min** | -124.350000 | 32.540000 | 1.000000 | 2.000000 | 1.00000 |
| **25%** | -121.800000 | 33.930000 | 18.000000 | 1447.750000 | 296.00000 |
| **50%** | -118.490000 | 34.260000 | 29.000000 | 2127.000000 | 435.00000 |
| **75%** | -118.010000 | 37.710000 | 37.000000 | 3148.000000 | 647.00000 |
| **max** | -114.310000 | 41.950000 | 52.000000 | 39320.000000 | 6445.00000 |

In [18]:
```
dataset.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 [19]:
```
dataset.columns
```

Out[19]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity'],
      dtype='object')
```

In [20]:
```
dataset.dropna(inplace = True)
```

In [21]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 20433 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20433 non-null  float64
 1   latitude            20433 non-null  float64
 2   housing_median_age  20433 non-null  float64
 3   total_rooms         20433 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20433 non-null  float64
 6   households          20433 non-null  float64
 7   median_income       20433 non-null  float64
 8   median_house_value  20433 non-null  float64
 9   ocean_proximity     20433 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.7+ MB
```

In [22]:
```
X = dataset.drop("median_house_value",axis=1)
y= dataset['median_house_value']
```

In [23]:
```
X
```

Out[23]:

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | populatic |
|---|---|---|---|---|---|---|
| **0** | -122.23 | 37.88 | 41.0 | 880.0 | 129.0 | 322 |
| **1** | -122.22 | 37.86 | 21.0 | 7099.0 | 1106.0 | 2401 |
| **2** | -122.24 | 37.85 | 52.0 | 1467.0 | 190.0 | 496 |
| **3** | -122.25 | 37.85 | 52.0 | 1274.0 | 235.0 | 558 |
| **4** | -122.25 | 37.85 | 52.0 | 1627.0 | 280.0 | 565 |
| **...** | ... | ... | ... | ... | ... | |
| **20635** | -121.09 | 39.48 | 25.0 | 1665.0 | 374.0 | 845 |
| **20636** | -121.21 | 39.49 | 18.0 | 697.0 | 150.0 | 356 |
| **20637** | -121.22 | 39.43 | 17.0 | 2254.0 | 485.0 | 1007 |
| **20638** | -121.32 | 39.43 | 18.0 | 1860.0 | 409.0 | 741 |
| **20639** | -121.24 | 39.37 | 16.0 | 2785.0 | 616.0 | 1387 |

20433 rows × 9 columns

In [24]:
```
y
```

Out[24]:
```
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: 20433, dtype: float64
```

In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
```

In [26]:
```
train_data = X_train.join(y_train)
```

In [27]:
```
train_data
```

Out[27]:

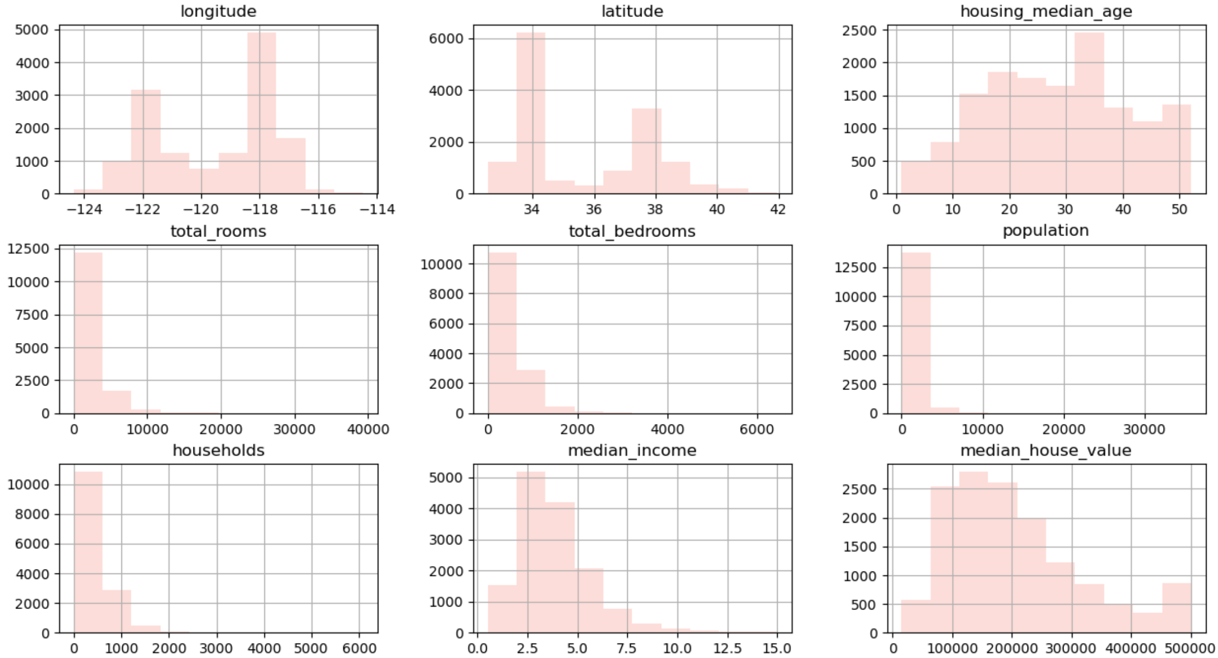
| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | populatio |
|---|---|---|---|---|---|---|
| **19566** | -120.96 | 37.61 | 23.0 | 3497.0 | 887.0 | 2467. |
| **7292** | -118.22 | 33.98 | 34.0 | 2225.0 | 753.0 | 2980. |
| **17618** | -121.94 | 37.28 | 27.0 | 2859.0 | 464.0 | 1144. |
| **17518** | -121.91 | 37.34 | 35.0 | 2189.0 | 607.0 | 1193. |
| **5172** | -118.28 | 33.95 | 41.0 | 835.0 | 208.0 | 707. |
| **...** | ... | ... | ... | ... | ... | . |
| **11397** | -117.97 | 33.72 | 24.0 | 2991.0 | 500.0 | 1437. |
| **12081** | -117.54 | 33.76 | 5.0 | 5846.0 | 1035.0 | 3258. |
| **5447** | -118.42 | 34.01 | 42.0 | 1594.0 | 369.0 | 952. |
| **866** | -122.04 | 37.57 | 12.0 | 5719.0 | 1064.0 | 3436. |
| **15948** | -122.43 | 37.73 | 52.0 | 3602.0 | 738.0 | 2270. |

14303 rows × 10 columns

In [28]:
```
sns.set_palette("RdPu")
```

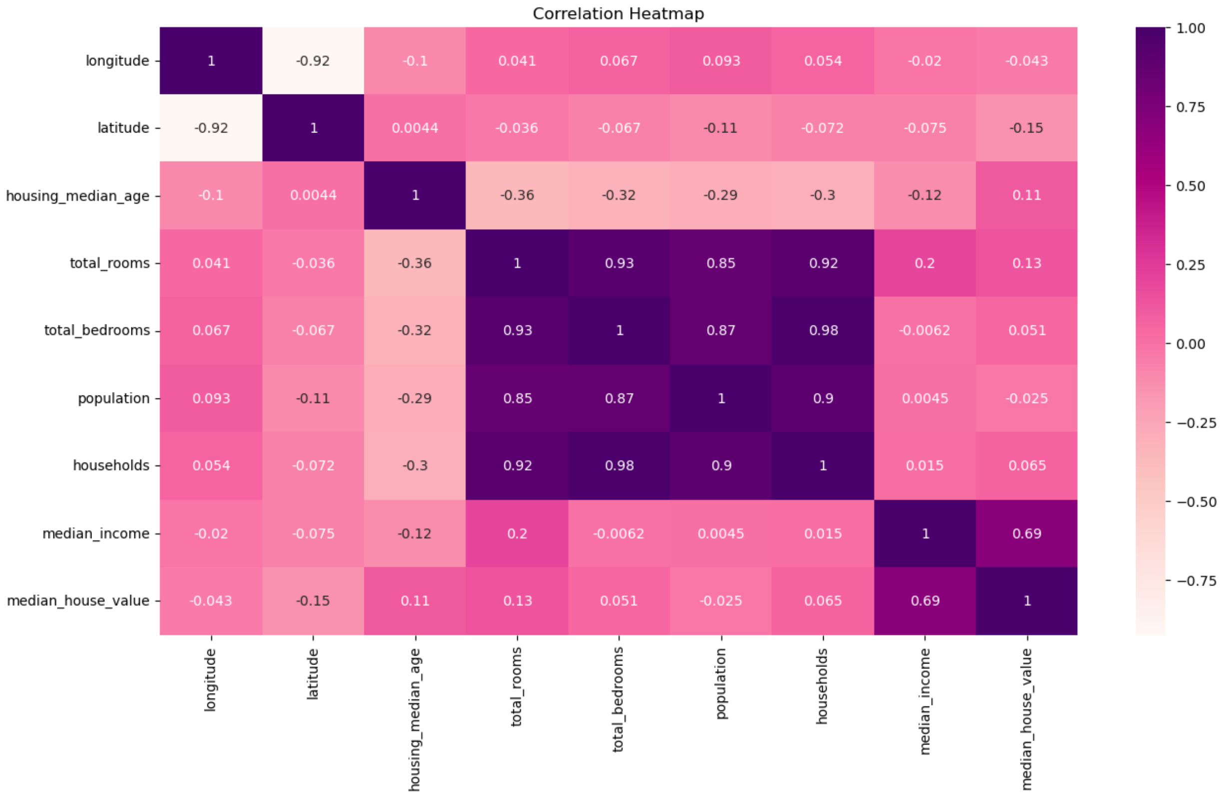
In [29]:
```
train_data.hist(figsize=(15,8))
```

Out[29]:
```
array([[<Axes: title={'center': 'longitude'}>,
        <Axes: title={'center': 'latitude'}>,
        <Axes: title={'center': 'housing_median_age'}>],
       [<Axes: title={'center': 'total_rooms'}>,
        <Axes: title={'center': 'total_bedrooms'}>,
        <Axes: title={'center': 'population'}>],
       [<Axes: title={'center': 'households'}>,
        <Axes: title={'center': 'median_income'}>,
        <Axes: title={'center': 'median_house_value'}>]], dtype=object)
```




In [30]:
```python
plt.figure(figsize=(15,8))
sns.heatmap(train_data.corr(numeric_only=True),annot = True, cmap="RdPu")
plt.title('Correlation Heatmap')
```

Out[30]:
```
Text(0.5, 1.0, 'Correlation Heatmap')
```



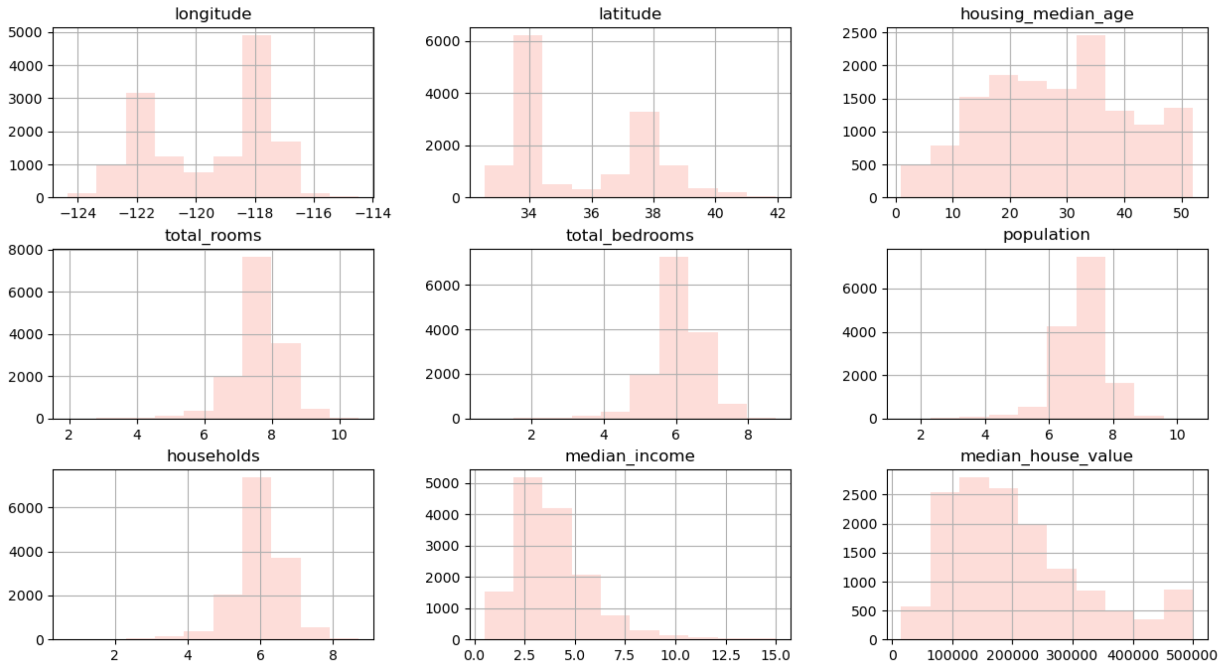

# PREPROCESSING

In [31]:
```
#To make see whats the distribution would look like (normal distributing)

train_data["total_rooms"] = np.log(train_data["total_rooms"] + 1)
train_data["total_bedrooms"] = np.log(train_data["total_bedrooms"] + 1)
train_data["population"] = np.log(train_data["population"] + 1)
train_data["households"] = np.log(train_data["households"] + 1)
```

In [32]:
```
train_data.hist(figsize=(15,8)) #check that data is bell curved
```

Out[32]:
```
array([[<Axes: title={'center': 'longitude'}>,
        <Axes: title={'center': 'latitude'}>,
        <Axes: title={'center': 'housing_median_age'}>],
       [<Axes: title={'center': 'total_rooms'}>,
        <Axes: title={'center': 'total_bedrooms'}>,
        <Axes: title={'center': 'population'}>],
       [<Axes: title={'center': 'households'}>,
        <Axes: title={'center': 'median_income'}>,
        <Axes: title={'center': 'median_house_value'}>]], dtype=object)
```




In [33]:
```
# making the ocean_proximity numerical since we assumed it might be effec
train_data = train_data.join(pd.get_dummies(train_data.ocean_proximity).a
```

In [34]:
```
train_data
```

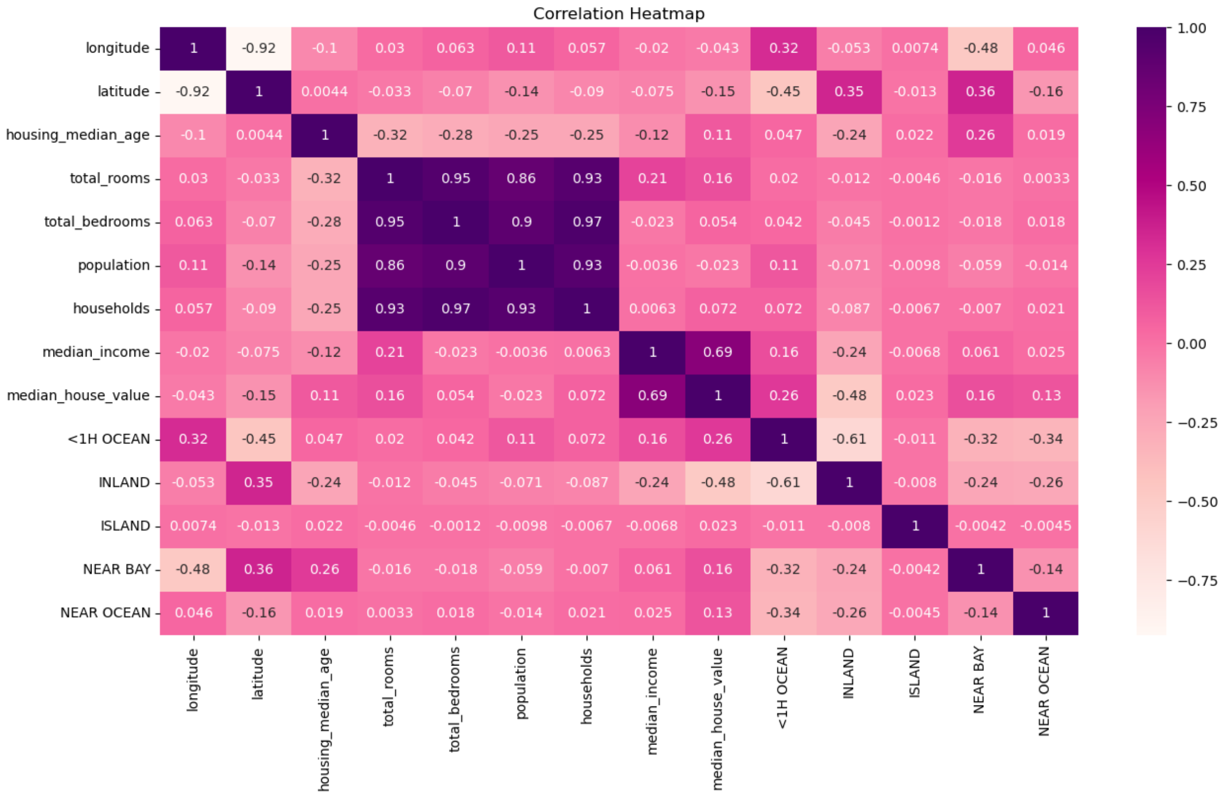
Out[34]:

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | populatio |
|---|---|---|---|---|---|---|
| **19566** | -120.96 | 37.61 | 23.0 | 8.159947 | 6.788972 | 7.81116 |
| **7292** | -118.22 | 33.98 | 34.0 | 7.707962 | 6.625392 | 8.00001 |
| **17618** | -121.94 | 37.28 | 27.0 | 7.958577 | 6.142037 | 7.04316 |
| **17518** | -121.91 | 37.34 | 35.0 | 7.691657 | 6.410175 | 7.08506 |
| **5172** | -118.28 | 33.95 | 41.0 | 6.728629 | 5.342334 | 6.56244 |
| **...** | ... | ... | ... | ... | ... | . |
| **11397** | -117.97 | 33.72 | 24.0 | 8.003697 | 6.216606 | 7.27100 |
| **12081** | -117.54 | 33.76 | 5.0 | 8.673684 | 6.943122 | 8.08917 |
| **5447** | -118.42 | 34.01 | 42.0 | 7.374629 | 5.913503 | 6.85961 |
| **866** | -122.04 | 37.57 | 12.0 | 8.651724 | 6.970730 | 8.14235 |
| **15948** | -122.43 | 37.73 | 52.0 | 8.189522 | 6.605298 | 7.72797 |

14303 rows × 14 columns

In [35]:
```
plt.figure(figsize=(15,8))
sns.heatmap(train_data.corr(numeric_only=True),annot = True, cmap="RdPu"
plt.title('Correlation Heatmap')
```

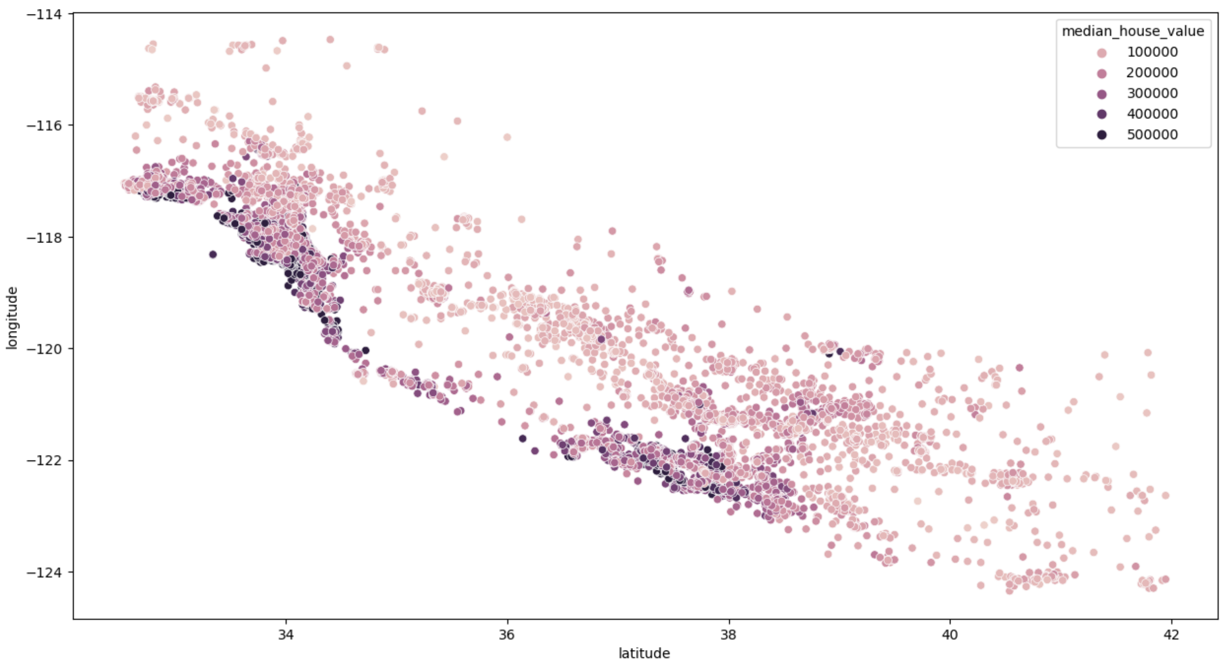
Out[35]:
```
Text(0.5, 1.0, 'Correlation Heatmap')
```




In [36]:

```
plt.figure(figsize=(15,8))
sns.scatterplot(x="latitude",y="longitude", data = train_data, hue="media
# you can see the houses closer to the coast are more expensive
```

Out[36]: <Axes: xlabel='latitude', ylabel='longitude'>

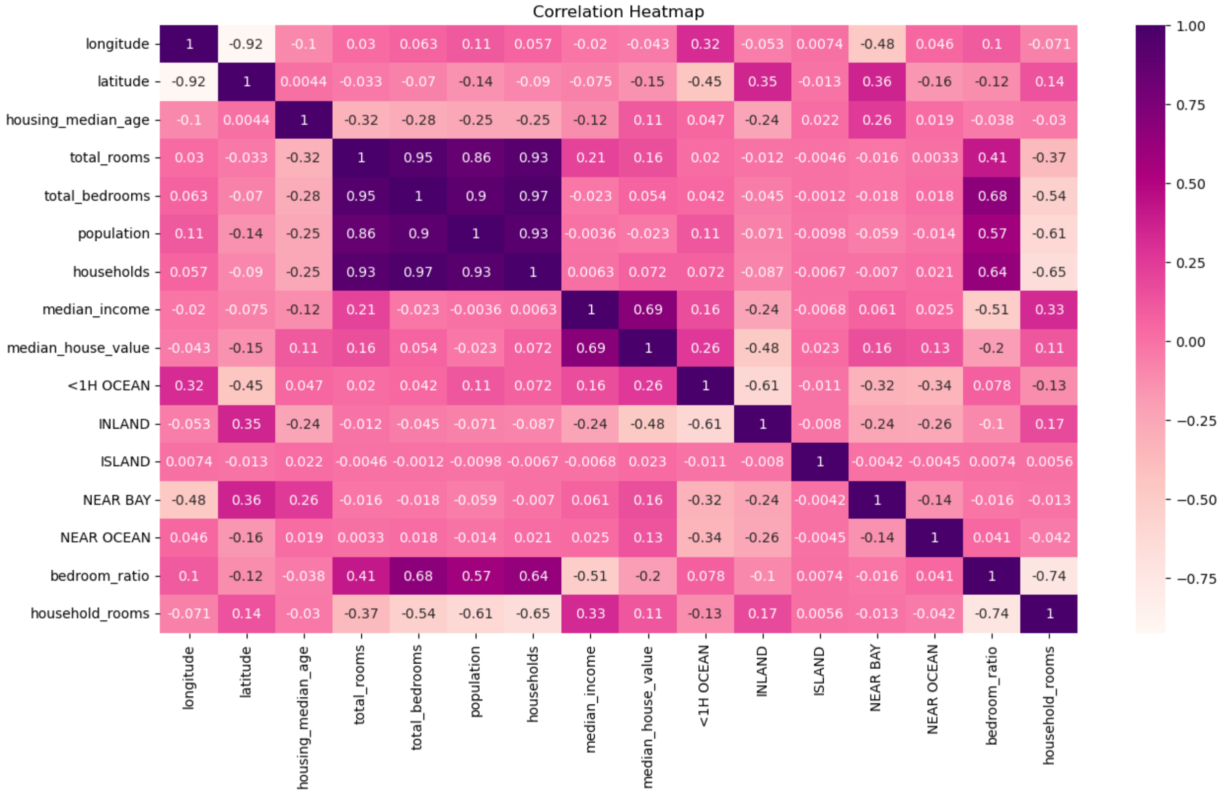



In [37]:

```
# adding features
train_data['bedroom_ratio'] = train_data['total_bedrooms']/train_data['to
train_data['household_rooms'] = train_data['total_rooms']/train_data['hou
```

In [38]:
```python
plt.figure(figsize=(15,8))
sns.heatmap(train_data.corr(numeric_only=True),annot = True, cmap="RdPu"
plt.title('Correlation Heatmap')
```

Out[38]: Text(0.5, 1.0, 'Correlation Heatmap')




# LINEAR REGRESSION MODEL

In [39]:
```python
from sklearn.linear_model import LinearRegression
X_train, y_train =train_data.drop("median_house_value",axis=1),train_data
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)

model = LinearRegression()
model.fit(X_train_s,y_train)
```

Out[39]:
```
▾ LinearRegression
LinearRegression()
```

In [40]:
```
# For test (not good practice)

test_data = X_test.join(y_test)
# To make see whats the distribution would look like (normal distributing

test_data["total_rooms"] = np.log(test_data["total_rooms"] + 1)
test_data["total_bedrooms"] = np.log(test_data["total_bedrooms"] + 1)
test_data["population"] = np.log(test_data["population"] + 1)
test_data["households"] = np.log(test_data["households"] + 1)

# making the ocean_proximity numerical since we assumed it might be effec
test_data = test_data.join(pd.get_dummies(test_data.ocean_proximity).asty

# adding features
test_data['bedroom_ratio'] = test_data['total_bedrooms']/test_data['total
test_data['household_rooms'] = test_data['total_rooms']/test_data['househ

X_test, y_test =test_data.drop("median_house_value",axis=1),test_data['me
X_test_s = scaler.transform(X_test)
```

In [41]:
```
y_pred = model.predict(X_test_s)
```

In [42]:
```
model.score(X_train_s, y_train)
```

Out[42]:
```
0.6690372201539236
```

In [43]:
```
model.score(X_test_s, y_test)
```

Out[43]:
```
0.6754321211145655
```

# LOSS-FUNCTION

In [44]:
```
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

In [45]:
```
print(f"Mean Squared Error: {mse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
Mean Squared Error: 4326792464.83
Mean Absolute Error: 47887.42
R-squared: 0.68
```

# LASSO REGULARIZATION

In [46]:
```
from sklearn import linear_model
from sklearn.linear_model import Lasso
scores_lasso = []
for alpha in [0.001, 0.01, 0.1, 1.0, 10.0, 20.0, 50.0]:
    lasso_reg = Lasso(alpha=alpha, max_iter=10, tol=0.1)
    lasso_reg.fit(X_train_s,y_train)
    y_pred_lasso = lasso_reg.predict(X_test_s)
    scores_lasso.append(lasso_reg.score(X_test_s, y_test))
```

In [47]:
```
scores_lasso
```

Out[47]:
```
[0.6654687907973387,
 0.6654687698633619,
 0.6654685605084898,
 0.6654664654495541,
 0.665445363838757,
 0.6654215955610625,
 0.6653482567691869]
```

# RIDGE REGULARIZATION

In [48]:
```
from sklearn.linear_model import Ridge
scores_rid = []
for alpha in [0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0, 100000.0, 1000000.0
    ridge_reg = Ridge(alpha=alpha, max_iter=10, tol=0.1)
    ridge_reg.fit(X_train_s,y_train)
    y_pred_reg = ridge_reg.predict(X_test_s)
    scores_rid.append(ridge_reg.score(X_test_s, y_test))
```

In [49]:
```
scores_rid
```

Out[49]:
```
[0.6754222403634237,
 0.6754255236430762,
 0.6752624216463456,
 0.6721370160243976,
 0.6520719410434335,
 0.5429964345391964,
 0.1970152588597247,
 0.025920766007146767]
```

# RandomizedSearchCV

In [50]:
```
param_grid = {'alpha': np.arange(0.0001, 1, 10),"solver": ['sag','lsqr']}
folds = KFold(n_splits = 7, shuffle = True, random_state = 100)
scores_cv = cross_val_score(model, X_train_s, y_train, scoring='r2', cv=f
scores_cv
```

Out[50]:
```
array([0.68098428, 0.64324138, 0.6704103 , 0.66233806, 0.6713499 ,
       0.6747685 , 0.66586176])
```

In [51]:
```
ridge1 = Ridge()
ridge_cv = RandomizedSearchCV(ridge1, param_grid, cv=folds, n_iter=5)
ridge_cv.fit(X_train_s, y_train)
print(ridge_cv.best_params_, ridge_cv.best_score_)
```

```
{'solver': 'lsqr', 'alpha': 0.0001} 0.6670306836073002
```

In [52]:
```
test_score_ridge = ridge_cv.score(X_test_s, y_test)
print(test_score_ridge)
```

```
0.6754216482443399
```

# GridSearchCV

In [53]:
```
# Set up the parameter grid
param_grid2 = {"alpha": np.linspace(0.00001, 1, 20)}

# Instantiate lasso_cv
lasso_cv = GridSearchCV(lasso_reg, param_grid2, cv=folds)

# Fit to the training data
lasso_cv.fit(X_train_s,y_train)
print("Tuned lasso paramaters: {}".format(lasso_cv.best_params_))
print("Tuned lasso score: {}".format(lasso_cv.best_score_))
```

```
Tuned lasso paramaters: {'alpha': 1e-05}
Tuned lasso score: 0.6600268107349938
```

In [54]:
```
test_score_lasso = lasso_cv.score(X_test_s, y_test)
print(test_score_lasso)
```

```
0.6654687931000596
```

# PLOTTING

In [55]:
```
pred_df=pd.DataFrame({'Actual Value':y_test,'Predicted Value':y_pred,'Dif
pred_df
```

Out[55]:

| | Actual Value | Predicted Value | Difference |
|---|---|---|---|
| **14416** | 245800.0 | 231067.322031 | 14732.677969 |
| **16383** | 137900.0 | 150445.454546 | -12545.454546 |
| **7731** | 218200.0 | 209348.366979 | 8851.633021 |
| **1410** | 220800.0 | 172476.224327 | 48323.775673 |
| **1335** | 170500.0 | 224936.315431 | -54436.315431 |
| **...** | ... | ... | ... |
| **16764** | 287700.0 | 306385.242972 | -18685.242972 |
| **5762** | 241900.0 | 219532.735526 | 22367.264474 |
| **12862** | 88400.0 | 135588.207330 | -47188.207330 |
| **18814** | 77500.0 | 37931.781431 | 39568.218569 |
| **12852** | 72900.0 | 70473.558956 | 2426.441044 |

6130 rows × 3 columns

# K-NN

In [56]:

```
from sklearn.neighbors import KNeighborsRegressor
```

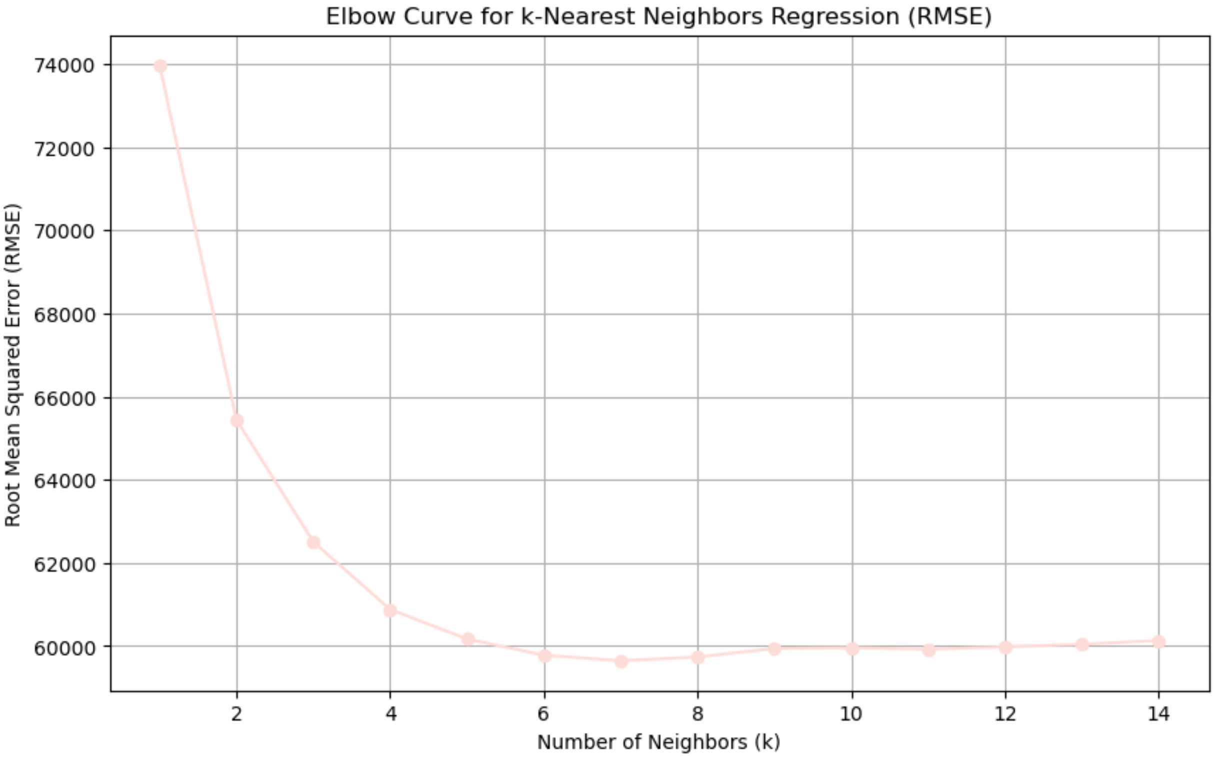
In [57]:

```
knn_rmses = []
for i in range(1,15):
    knn = KNeighborsRegressor(n_neighbors=i)
    knn.fit(X_train_s,y_train)
    y_pred = knn.predict(X_test_s)
    knn_rmse=(mean_squared_error(y_test,y_pred))**0.5
    knn_r2=r2_score(y_test,y_pred)
    knn_rmses.append(knn_rmse)
```

# ELBOW CURVE

In [58]:

```
plt.figure(figsize=(10, 6))
plt.plot(range(1, 15), knn_rmses, marker='o')
plt.title('Elbow Curve for k-Nearest Neighbors Regression (RMSE)')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Root Mean Squared Error (RMSE)')
plt.grid(True)
plt.show()
```




# GRID SEARCH KNN

In [59]:
```
param_grid = {'n_neighbors': range(1, 30)}


# Perform grid search with cross-validation
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X_train_s, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Print the best hyperparameters
print("Best Hyperparameters:", best_params)

# Get the best model
best_knn = grid_search.best_estimator_

# Make predictions on the test set
y_pred_grid = best_knn.predict(X_test_s)
```

```
Best Hyperparameters: {'n_neighbors': 11}
```

In [60]:
```
best_knn.score(X_test_s, y_test)
```

Out[60]:
```
0.7306482377570549
```

# RANDOMIZED SEARCH KNN

In [61]:
```
from scipy.stats import randint

# Define the parameter distribution
param_dist = {'n_neighbors': randint(1, 30)}

# Perform randomized search with cross-validation
random_search = RandomizedSearchCV(knn, param_distributions=param_dist, n
random_search.fit(X_train_s, y_train)

# Get the best hyperparameters
best_params_random = random_search.best_params_

# Print the best hyperparameters
print("Best Hyperparameters (Randomized Search):", best_params_random)

# Get the best model
best_knn_random = random_search.best_estimator_

# Make predictions on the test set
y_pred_random = best_knn_random.predict(X_test_s)
```

```
Best Hyperparameters (Randomized Search): {'n_neighbors': 8}
```

In [62]:
```
best_knn_random.score(X_test_s, y_test)
```

Out[62]:
```
0.7323180432594305
```

# CROSS-VALIDATION

In [63]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import RandomizedSearchCV
```

In [64]:
```
from sklearn.model_selection import cross_validate

# Assuming model is your regression model, and best_knn_random is your KN
folds = KFold(n_splits=7, shuffle=True, random_state=100)

# For the linear regression model
scoring_lr = {'r2': 'r2', 'neg_mean_squared_error': 'neg_mean_squared_err
cv_results_lr = cross_validate(model, X_train_s, y_train, scoring=scoring

# For the KNN model
scoring_knn = {'r2': 'r2', 'neg_mean_squared_error': 'neg_mean_squared_er
cv_results_knn = cross_validate(best_knn_random, X_train_s, y_train, scor

# Extract the scores
scores_lr_r2 = cv_results_lr['test_r2']
scores_lr_mse = cv_results_lr['test_neg_mean_squared_error']

scores_knn_r2 = cv_results_knn['test_r2']
scores_knn_mse = cv_results_knn['test_neg_mean_squared_error']
```

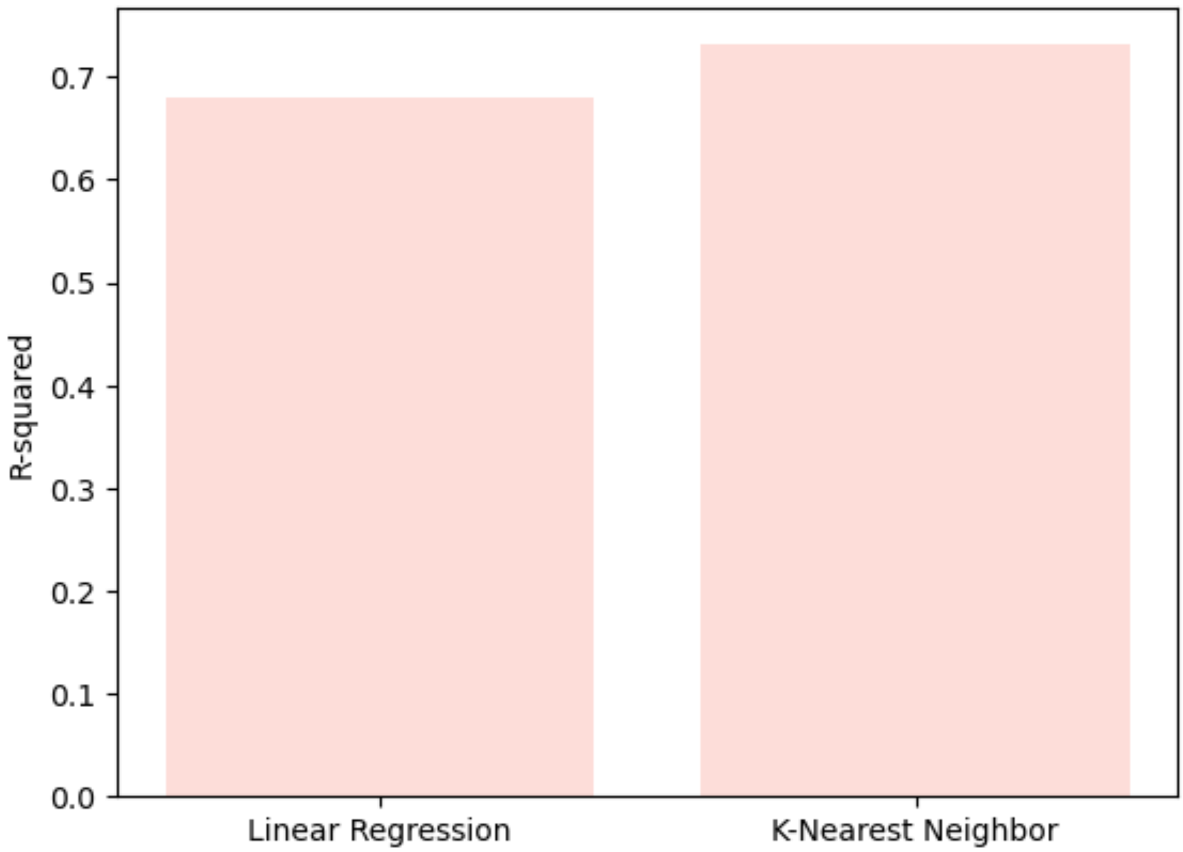
In [67]:
```
best_lr_r2 = np.max(scores_lr_r2)
lr_rmse = -np.max(scores_lr_mse)
best_knn_r2 = np.max(scores_knn_r2)
knn_rmse = -np.max(scores_knn_mse)
```

In [68]:
```
print("Best R-squared in Linea:r" ,best_lr_r2)
print("Best RMSE in Linear:" ,lr_rmse)
print("Best R-squared in KNN:" ,best_knn_r2)
print("Best RMSE in KNN:" ,knn_rmse)
```

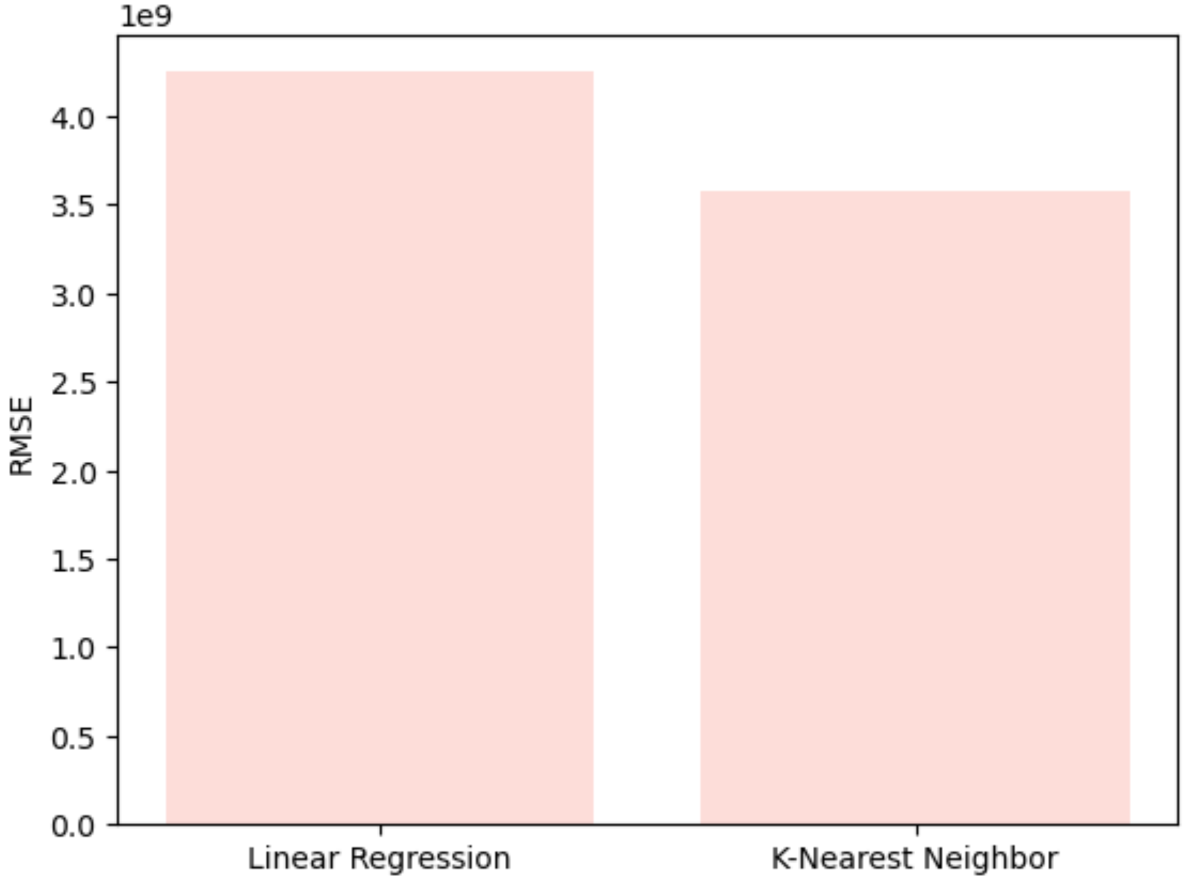
```
Best R-squared in Linea:r 0.6809842804858977
Best RMSE in Linear: 4249601652.501362
Best R-squared in KNN: 0.7313163279751095
Best RMSE in KNN: 3579129512.4146204
```

# PLOTTING TWO MODELS

In [69]:
```
plt.bar(x=['Linear Regression' , 'K-Nearest Neighbor'] , height=[best_lr_
plt.ylabel("R-squared")
plt.show()
```




In [70]:
```
plt.bar(x=['Linear Regression' , 'K-Nearest Neighbor'] , height=[lr_rmse,
plt.ylabel("RMSE")
plt.show() #shorter better
```


1e9
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
RMSE
Linear Regression
K-Nearest Neighbor

In [ ]: