DATASET DOWNLOAD FROM: https://www.kaggle.com/datasets/camnugent/california-housing-prices

```
from sklearn.datasets import fetch_california_housing
import pandas as pd

# Load dataset
housing = fetch_california_housing(as_frame=True)
# Convert to DataFrame
df = housing.frame

# Save to CSV
df.to_csv("california_housing.csv", index=False)

print("✅ california_housing.csv saved successfully!")
print(df.head())
```

```
✅ california_housing.csv saved successfully!
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude  MedHouseVal
0    -122.23        4.526
1    -122.22        3.585
2    -122.24        3.521
3    -122.25        3.413
4    -122.25        3.422
```

```
df.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVa
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.00000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655	35.631861	-119.569704	2.06855
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050	2.135952	2.003532	1.15395
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308	32.540000	-124.350000	0.14999
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741	33.930000	-121.800000	1.19600
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116	34.260000	-118.490000	1.79700
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261	37.710000	-118.010000	2.64725
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333	41.950000	-114.310000	5.00001

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   MedInc       20640 non-null  float64
 1   HouseAge     20640 non-null  float64
 2   AveRooms     20640 non-null  float64
 3   AveBedrms    20640 non-null  float64
 4   Population   20640 non-null  float64
 5   AveOccup     20640 non-null  float64
 6   Latitude     20640 non-null  float64
 7   Longitude    20640 non-null  float64
 8   MedHouseVal  20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

```
df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

```
df.tail()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09	0.781
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21	0.771
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22	0.923
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32	0.847
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24	0.894

```
df.tail(10)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
20630	3.5673	11.0	5.932584	1.134831	1257.0	2.824719	39.29	-121.32	1.120
20631	3.5179	15.0	6.145833	1.141204	1200.0	2.777778	39.33	-121.40	1.072
20632	3.1250	15.0	6.023377	1.080519	1047.0	2.719481	39.26	-121.45	1.156
20633	2.5495	27.0	5.445026	1.078534	1082.0	2.832461	39.19	-121.53	0.983
20634	3.7125	28.0	6.779070	1.148256	1041.0	3.026163	39.27	-121.56	1.168
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09	0.781
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21	0.771
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22	0.923
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32	0.847
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24	0.894

Step 1: Import libraries

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
```

Step 2: Load Dataset

```
housing = fetch_california_housing(as_frame=True)
df = housing.frame
```

```
# Features & target
X = df.drop("MedHouseVal", axis=1)
y = df["MedHouseVal"]
```

```
# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state = 42)
```

```
# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Step 3: Train Multiple Models

```
# Define models
models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(random_state=42, n_estimators=100),
    "Gradient Boosting": GradientBoostingRegressor(random_state=42),
    "KNN": KNeighborsRegressor(n_neighbors=5)
}

# Store results
results = []

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    results.append({
        "Model": name,
        "MAE": mae,
        "RMSE": rmse,
        "R²": r2
    })
```

Step 4: Compare Models

```
# Convert results to DataFrame
results_df = pd.DataFrame(results).sort_values(by="R²", ascending=False)
print(results_df)
```

```
               Model       MAE      RMSE        R²
2      Random Forest  0.327425  0.505143  0.805275
3  Gradient Boosting  0.371650  0.542217  0.775643
4                KNN  0.446154  0.657588  0.670010
1      Decision Tree  0.453904  0.702829  0.623042
0  Linear Regression  0.533200  0.745581  0.575788
```

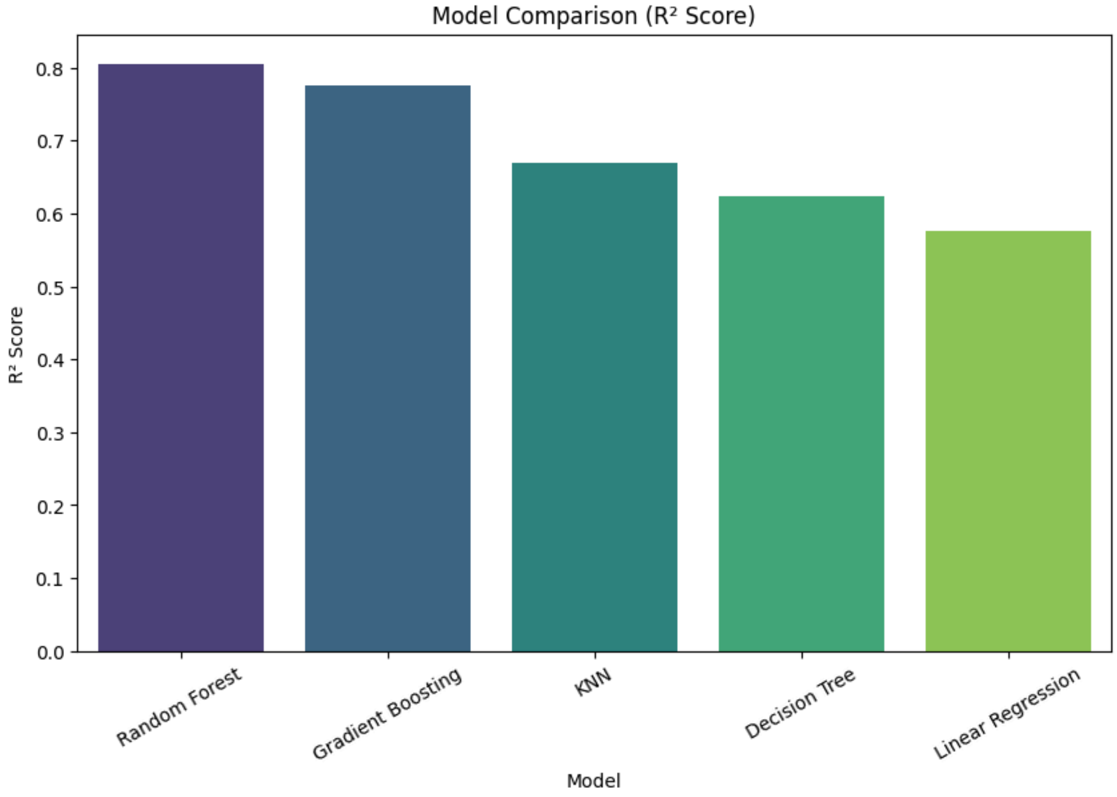
Step 5: Visualization

```
# Plot comparison
plt.figure(figsize=(10,6))
sns.barplot(data=results_df, x="Model", y="R²", palette="viridis")
plt.title("Model Comparison (R² Score)")
plt.ylabel("R² Score")
plt.xticks(rotation=30)
plt.show()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_14356\718367510.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se

  sns.barplot(data=results_df, x="Model", y="R²", palette="viridis")
```



Step 6: Save the Best Model

```
import joblib

models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest": RandomForestRegressor(),
    "Gradient Boosting": GradientBoostingRegressor(),
    "KNN": KNeighborsRegressor()
}

# Train and evaluate models
scores = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    scores[name] = r2_score(y_test, y_pred)

# Find best model
best_model_name = max(scores, key=scores.get)
best_model = models[best_model_name]

print(f"✅ Best model is: {best_model_name} with R² = {scores[best_model_name]:.4f}")

# Save best model
joblib.dump(best_model, "best_model.pkl")
print("📁 Best model saved as best_model.pkl")
```

```
✅ Best model is: Random Forest with R² = 0.8054
📁 Best model saved as best_model.pkl
```

```
# Load best model
loaded_model = joblib.load("best_model.pkl")

# Take user input
print("\nEnter values for prediction:")
MedInc = float(input("Median Income: "))
HouseAge = float(input("House Age: "))
AveRooms = float(input("Average Rooms: "))
```

```
AveBedrms = float(input("Average Bedrooms: "))
Population = float(input("Population: "))
AveOccup = float(input("Average Occupancy: "))
Latitude = float(input("Latitude: "))
Longitude = float(input("Longitude: "))

# Store inputs in a dict for display
user_inputs = {
    "Median Income": MedInc,
    "House Age": HouseAge,
    "Average Rooms": AveRooms,
    "Average Bedrooms": AveBedrms,
    "Population": Population,
    "Average Occupancy": AveOccup,
    "Latitude": Latitude,
    "Longitude": Longitude
}

# Display entered values
print("\n📌 Entered Input Values:")
for feature, value in user_inputs.items():
    print(f"{feature}: {value}")

# Create input array
user_input = np.array([[MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup, Latitude, Longitude]])

# Predict
prediction = loaded_model.predict(user_input)

# Multiply by 100,000 to convert to actual USD price
price_usd = prediction[0] * 100000
print(f"\n🏠 Predicted House Price: ${price_usd:,.2f} USD")
```

```
Enter values for prediction:

📌 Entered Input Values:
Median Income: 5.0
House Age: 24.0
Average Rooms: 4.0
Average Bedrooms: 2.0
Population: 2000.0
Average Occupancy: 3.4
Latitude: 32.6
Longitude: -119.3

🏠 Predicted House Price: $369,515.42 USD
```

```
# Take user input
print("\nEnter values for prediction:")
MedInc = float(input("Median Income: "))
HouseAge = float(input("House Age: "))
AveRooms = float(input("Average Rooms: "))
AveBedrms = float(input("Average Bedrooms: "))
Population = float(input("Population: "))
AveOccup = float(input("Average Occupancy: "))
Latitude = float(input("Latitude: "))
Longitude = float(input("Longitude: "))

# Store inputs in a dict for display
user_inputs = {
    "Median Income": MedInc,
    "House Age": HouseAge,
    "Average Rooms": AveRooms,
    "Average Bedrooms": AveBedrms,
    "Population": Population,
    "Average Occupancy": AveOccup,
    "Latitude": Latitude,
    "Longitude": Longitude
}

# Display entered values
print("\n📌 Entered Input Values:")
for feature, value in user_inputs.items():
    print(f"{feature}: {value}")

# Create input array
user_input = np.array([[MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup, Latitude, Longitude]])

# Predict
prediction = loaded_model.predict(user_input)

# Multiply by 100,000 to convert to actual USD price
```

```
price_usd = prediction[0] * 100000
print(f"\n🏠 Predicted House Price: ${price_usd:,.2f} USD")
```

```
Enter values for prediction:

📌 Entered Input Values:
Median Income: 3.0
House Age: 29.0
Average Rooms: 5.0
Average Bedrooms: 7.0
Population: 20000.0
Average Occupancy: 3.7
Latitude: 45.5
Longitude: -78.23

🏠 Predicted House Price: $355,576.26 USD
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```