Importing necessary Libraries

In [ ]:
```
import pandas as pd
import numpy as np
import sympy as sy
import scipy as sp
import matplotlib.pyplot as plt
```

Fetching and Loading the Data into DataFrame

In [ ]:
```
from sklearn.datasets import fetch_california_housing

# Fetch the California housing dataset
housing = fetch_california_housing()

# Create a DataFrame from the data
df = pd.DataFrame(housing.data, columns=housing.feature_names)

# Add the target variable to the DataFrame
df['MedHouseVal'] = housing.target

# Display the first few rows of the DataFrame
print(df.head())
```

```
   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
```

Data Exploration

Now we'll perform some commands to explore our data set

1. Basic Information

- shape of the Data

In [ ]:
```
df.shape
```

Out[ ]: (20640, 9)

- column names

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
       'Latitude', 'Longitude', 'MedHouseVal'],
      dtype='object')
```

- datatypes

In [ ]:
```
df.dtypes
```

Out[ ]:
```
MedInc         float64
HouseAge       float64
AveRooms       float64
AveBedrms      float64
Population     float64
AveOccup       float64
Latitude       float64
Longitude      float64
MedHouseVal    float64
dtype: object
```

- some statistical information

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

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333

2. Checking for missing Values

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

Out[ ]:
```
MedInc         0
HouseAge       0
AveRooms       0
AveBedrms      0
Population     0
AveOccup       0
Latitude       0
Longitude      0
MedHouseVal    0
dtype: int64
```

3. Value count for specific columns

In [ ]:
```
df['MedHouseVal'].value_counts()
```

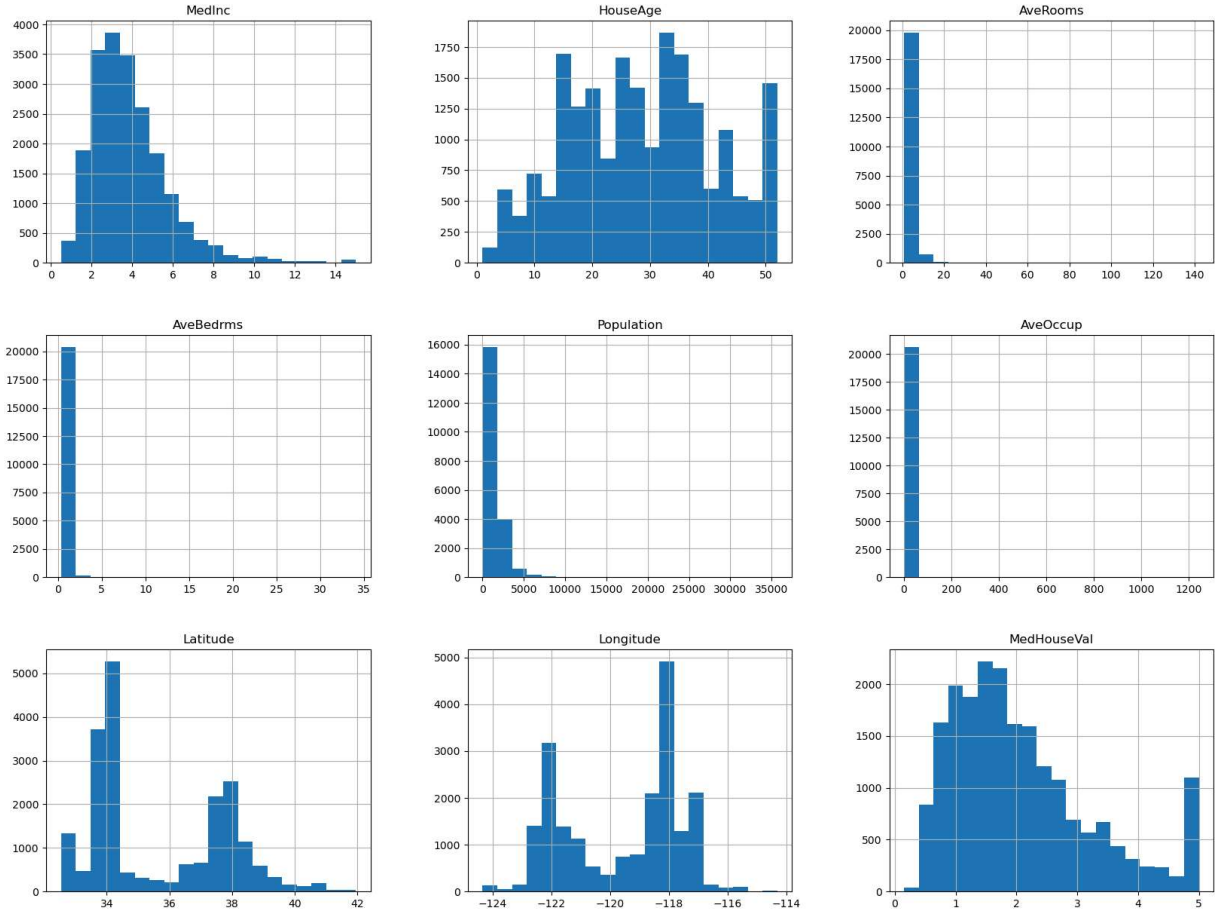
Out[ ]:
```
MedHouseVal
5.00001    965
1.37500    122
1.62500    117
1.12500    103
1.87500     93
          ...
3.59200      1
0.54900      1
3.77600      1
0.81200      1
0.47000      1
Name: count, Length: 3842, dtype: int64
```

Visualizing the Data

libraries like matplotlib and seaborn allows for advanced visualizations.

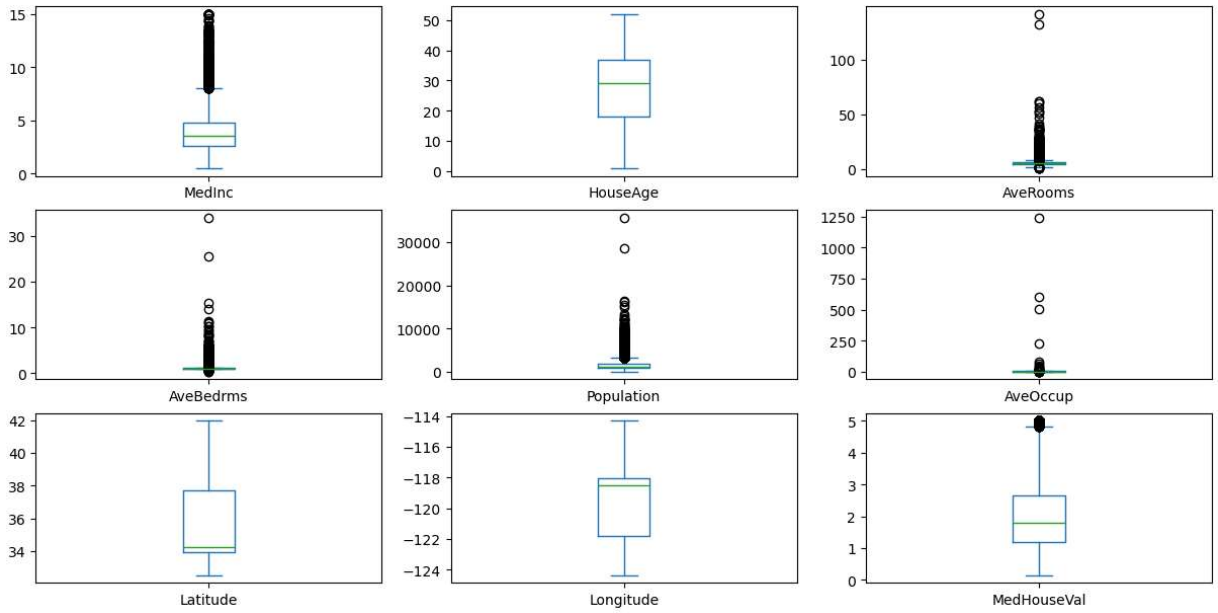
1. Histograms of Each Feature:

In [ ]:
```
import matplotlib.pyplot as plt

df.hist(bins=20, figsize=(20, 15))
plt.show()
```



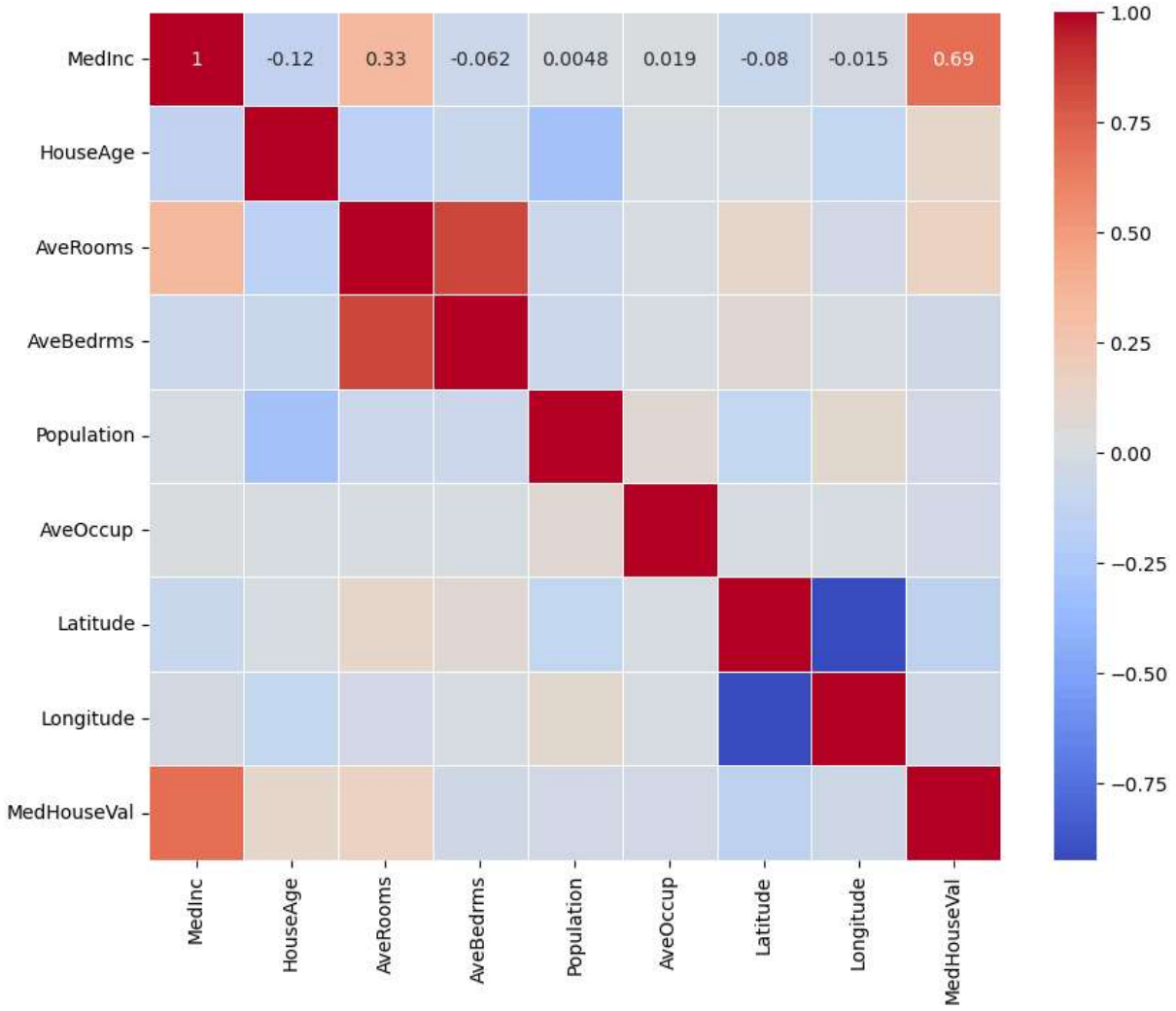
2. Boxplots to Identify Outliers:

```
In [ ]: df.plot(kind='box', subplots=True, layout=(4,3), figsize=(15, 10), sharex=False, sh
        plt.show()
```



3. Correlation Matrix and Heatmap:

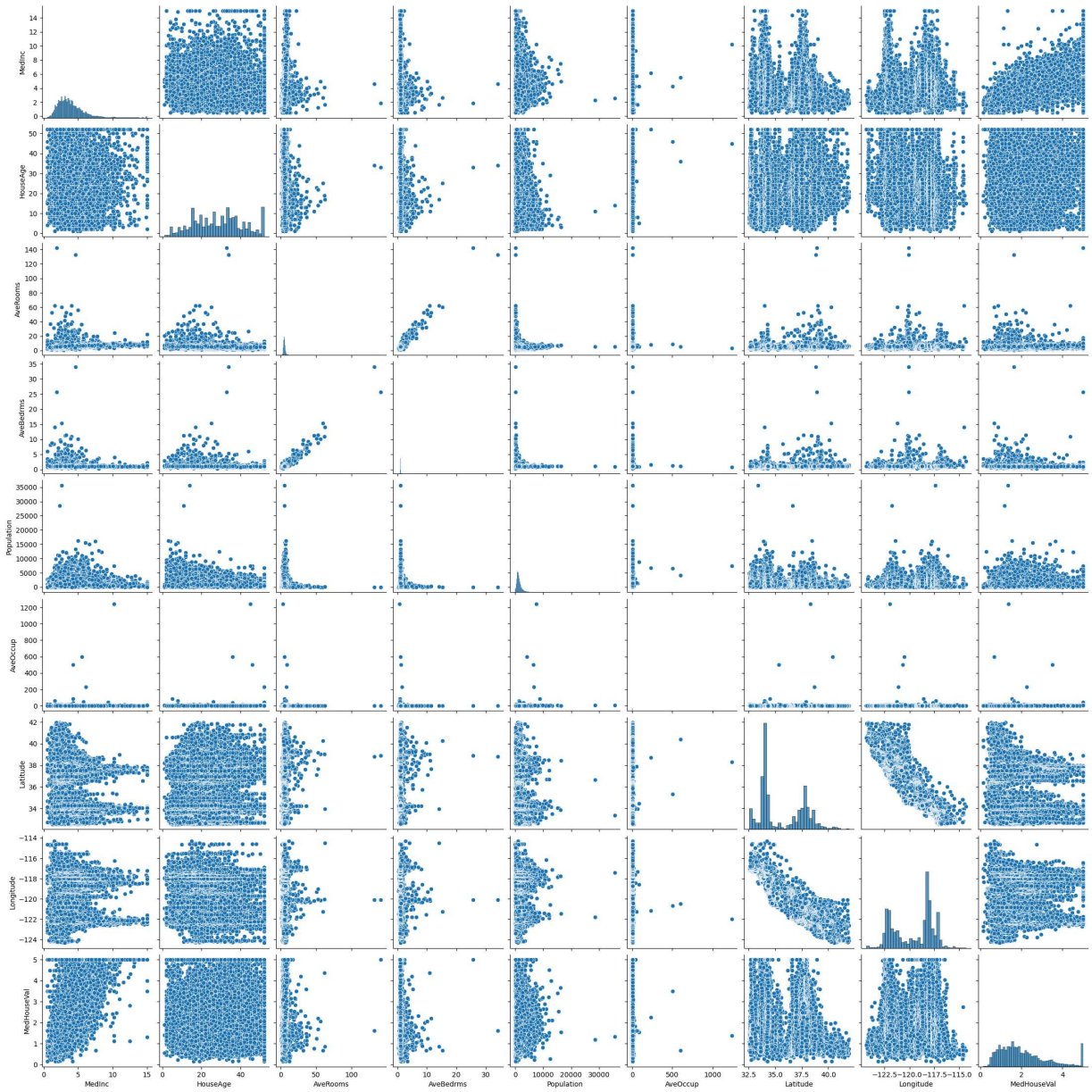
In [ ]:
```
import seaborn as sns

correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.show()
```



4. Pairplot for Visualizing Relationships:

In [ ]:
```
sns.pairplot(df)
plt.show()
```

```
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Additional Analysis

1. Grouping and Aggregation:

In [ ]:
```
print(df.groupby('MedHouseVal').mean())
```

```
                MedInc   HouseAge  AveRooms  AveBedrms  Population  AveOccup  \
MedHouseVal
0.14999       2.122475  30.750000  6.575951   2.016259   305.25000  2.566440
0.17500       2.366700  39.000000  3.572464   1.217391   259.00000  1.876812
0.22500       1.818075  36.250000  3.975628   1.265805  2112.00000  3.652335
0.25000       0.857100  21.000000  1.629630   1.222222    64.00000  2.370370
0.26600       2.301300  34.000000  4.897959   1.051020   808.00000  2.748299
...                ...        ...       ...        ...         ...       ...
4.98800       8.248000  29.000000  7.072727   0.978182   826.00000  3.003636
4.99000       8.148900  18.000000  6.600817   1.001362  1634.00000  2.226158
4.99100       6.786100  28.000000  7.386861   1.083942   617.00000  2.251825
5.00000       3.899581  38.000000  4.773400   1.094456  1036.00000  2.097639
5.00001       7.825123  33.802073  6.817436   1.097833  1112.80829  2.570442

                Latitude   Longitude
MedHouseVal
0.14999       37.665000 -120.197500
0.17500       34.150000 -118.330000
0.22500       36.005000 -119.335000
0.25000       32.790000 -114.650000
0.26600       35.130000 -119.450000
...                 ...         ...
4.98800       37.330000 -122.060000
4.99000       37.890000 -122.180000
4.99100       33.550000 -117.770000
5.00000       35.584444 -120.155556
5.00001       35.225751 -119.702477

[3842 rows x 8 columns]
```

2. Filtering Data:

In [ ]:
```
filtered_df = df[df['MedHouseVal'] > 2.0]
print(filtered_df.head())
```

```
   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
```

3. Sorting Data:

In [ ]:
```
sorted_df = df.sort_values(by='MedHouseVal', ascending=False)
print(sorted_df.head())
```

```
        MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
10667  10.1275      27.0  7.493421   1.039474       772.0  2.539474     33.55
16916   7.0422      52.0  7.045455   1.090909       967.0  2.585561     37.57
16946   6.1349      33.0  6.893417   0.978056       827.0  2.592476     37.55
8877    8.0257      48.0  7.781046   1.127451       859.0  2.807190     34.04
8878   15.0001      52.0  8.000000   0.997333      1143.0  3.048000     34.04

       Longitude  MedHouseVal
10667    -117.88      5.00001
16916    -122.34      5.00001
16946    -122.33      5.00001
8877     -118.49      5.00001
8878     -118.50      5.00001
```

Data Preprocessing

1. Splitting the Data

Split the dataset into training and testing sets to evaluate the model's performance on unseen data.

In [ ]:
```python
from sklearn.model_selection import train_test_split

X = df.drop('MedHouseVal', axis=1)  # Features
Y = df['MedHouseVal']  # Target variable

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_sta
```

2. Feature Scaling:

In [ ]:
```python
# Normalize or standardize the feature values to ensure that they are on a similar
# This can help improve the performance of the model.

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

3. Feature Engineering:

In [ ]:
```python
# Create new features or modify existing ones to improve the model's performance.

df['RoomsPerHousehold'] = df['AveRooms'] / df['AveOccup']
df['BedroomsPerRoom'] = df['AveBedrms'] / df['AveRooms']
df['PopulationPerHousehold'] = df['Population'] / df['AveOccup']
```

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

Out[ ]:

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

Training Machine Learning Models

1. Training Linear Regression Model

In [ ]:
```
from sklearn.linear_model import LinearRegression
linear = LinearRegression()
linear.fit(X_train, Y_train)
print(linear.coef_)
```

```
[ 0.85438303  0.12254624 -0.29441013  0.33925949 -0.00230772 -0.0408291
 -0.89692888 -0.86984178]
```

2. Training Random Forest Model

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
rf_regressor = RandomForestRegressor()
rf_regressor.fit(X_train, Y_train)
```

Out[ ]:
```
▾ RandomForestRegressor

RandomForestRegressor()
```

3. Training Decision Tree Regressor

In [ ]:
```
from sklearn.tree import DecisionTreeRegressor
dt_regressor = DecisionTreeRegressor()
dt_regressor.fit(X_train, Y_train)
```

Out[ ]:
```
▾ DecisionTreeRegressor

DecisionTreeRegressor()
```

4. Training Multilayer Preceptron

In [ ]:
```
from sklearn.neural_network import MLPRegressor
mlp_regressor = MLPRegressor(hidden_layer_sizes=(50,), activation='relu', solver='a
mlp_regressor.fit(X_train, Y_train)
```

Out[ ]:
```
MLPRegressor
MLPRegressor(hidden_layer_sizes=(50,), max_iter=1000)
```

Evaluating Machine Learning Models

In [ ]:
```
# Evaluate each model's performance on the test set using appropriate metrics,
# such as Mean Squared Error (MSE)

from sklearn.metrics import mean_squared_error
```

1. Evaluating Linear Regression Model

In [ ]:
```
Y_linear_pred = linear.predict(X_test)
linear_mse = mean_squared_error(Y_test, Y_linear_pred)
print(f'Linear Regression MSE: {linear_mse}')
```

```
Linear Regression MSE: 0.5558915986952443
```

2. Evaluating Random Forest

In [ ]:
```
Y_rf_pred = rf_regressor.predict(X_test)
rf_mse = mean_squared_error(Y_test, Y_rf_pred)
print(f"Random Forest MSE: {rf_mse}")
```

```
Random Forest MSE: 0.25386868848241767
```

3. Evaluating Decision Tree Regressor

In [ ]:
```
Y_dt_pred = dt_regressor.predict(X_test)
dt_mse = mean_squared_error(Y_test, Y_dt_pred)
print(f"Decision Tree MSE: {dt_mse}")
```

```
Decision Tree MSE: 0.5025298174847869
```

4. Evaluating Multilayer Preceptron

In [ ]:
```
Y_mlp_pred = mlp_regressor.predict(X_test)
mlp_mse = mean_squared_error(Y_test, Y_mlp_pred)
print(f"Multilayer Perceptron: {mlp_mse}")
```

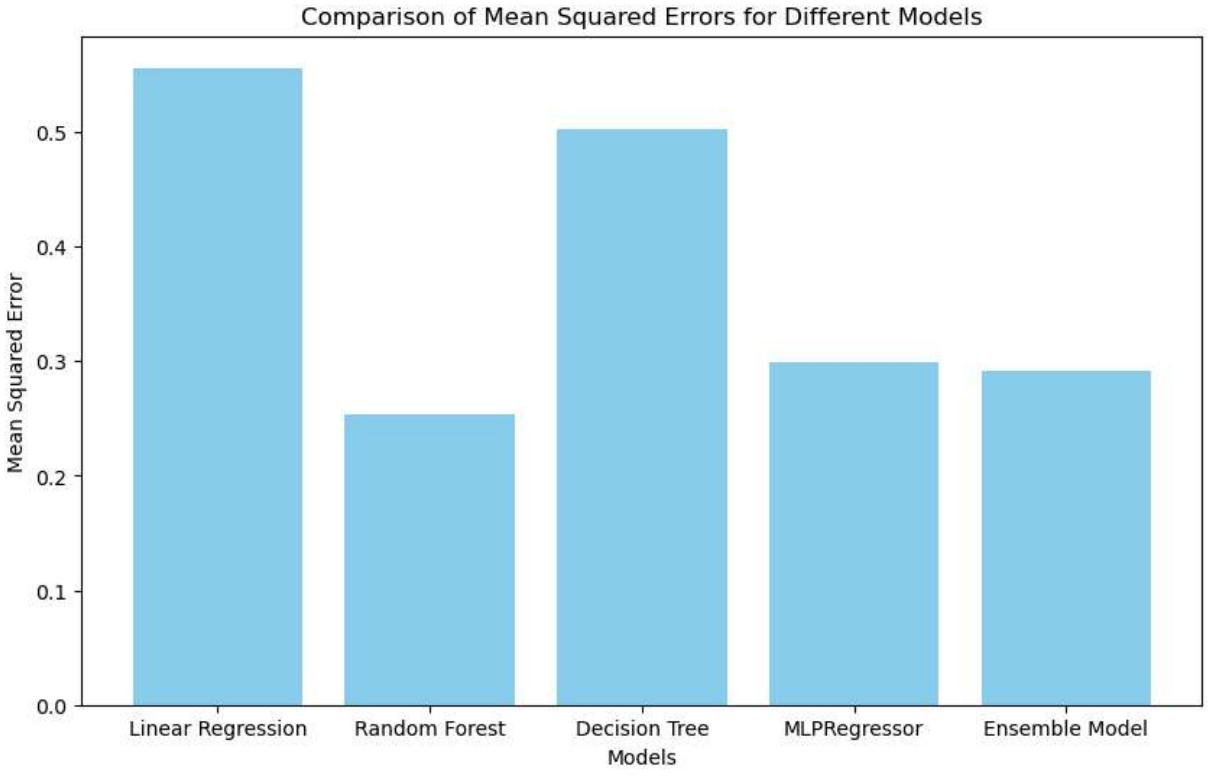
```
Multilayer Perceptron: 0.29918381889078527
```

Ensembling the Architecture

In [ ]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, Voti
from sklearn.metrics import mean_squared_error


linear_reg = LinearRegression()
decision_tree = DecisionTreeRegressor(random_state=42)
random_forest = RandomForestRegressor(n_estimators=100, random_state=42)
gradient_boost = GradientBoostingRegressor(random_state=42)


ensemble_model = VotingRegressor(estimators=[
    ('lr', linear_reg),
    ('dt', decision_tree),
    ('rf', random_forest),
    ('gb', gradient_boost)
])


ensemble_model.fit(X_train, Y_train)

y_ensemble_pred = ensemble_model.predict(X_test)


ensemble_mse = mean_squared_error(Y_test, y_ensemble_pred)
rmse = np.sqrt(ensemble_mse)
print(f"Mean Squared Error: {ensemble_mse}")
print(f"Root Mean Squared Error: {rmse}")
```

```
Mean Squared Error: 0.29162168990781495
Root Mean Squared Error: 0.5400200828745306
```

In [ ]:
```
import matplotlib.pyplot as plt
models = ["Linear Regression", "Random Forest", "Decision Tree", "MLPRegressor", "E
mse_values = [linear_mse, rf_mse, dt_mse, mlp_mse, ensemble_mse]

plt.figure(figsize=(10, 6))
plt.bar(models, mse_values, color='skyblue')
plt.xlabel("Models")
plt.ylabel("Mean Squared Error")
plt.title("Comparison of Mean Squared Errors for Different Models")
plt.show()
```


Comparison of Mean Squared Errors for Different Models
Mean Squared Error
0.0
0.1
0.2
0.3
0.4
0.5
Linear Regression
Random Forest
Decision Tree
MLPRegressor
Ensemble Model
Models

In [ ]: