Linear Regression Tutorial

Creator: Muhammad Bilal Alam

What is Linear Regression?

Regression analysis is a statistical method that helps us understand the relationship between two or more variables. It is a technique used to find the correlation between variables and how they might influence each other.

In simple terms, regression analysis helps us predict how one variable may change based on changes in another variable. For example, if we want to predict how much someone might weigh based on their height, regression analysis can help us make that prediction.

The method involves fitting a mathematical model to the data and then using that model to make predictions about future or unknown data points. Regression analysis is commonly used in many fields such as finance, economics, and social sciences to help us better understand the relationship between variables and make predictions about future outcomes.

Regression Equation:

The regression equation represents the mathematical relationship between the independent variable(s) and the dependent variable in a regression analysis. The general form of a simple linear regression equation is:

$$\mathbf{y = \beta_0 + \beta_1 x_1 + \varepsilon}$$

Where:

- y is the dependent variable (the one being predicted or explained)
- x_1 is the independent variable (the predictor variable)
- β_0 is the intercept or constant term
- β_1 is the slope or coefficient for the independent variable
- ε is the error term (represents the random variation or noise in the data)

The values of β_0 and β_1 are estimated using the regression analysis, and the resulting equation can be used to predict the value of y for any given value of x_1. This equation can be extended to multiple linear regression when more than one independent variable is included in the model.

Types of Regressions:

In data science, there are several types of linear regression models, including:

- **Simple linear regression:** This type of regression involves one independent variable and one dependent variable, and the relationship between them is linear.

- **Multiple linear regression:** This type of regression involves two or more independent variables and one dependent variable, and the relationship between them is still linear.
- **Polynomial regression:** This type of regression involves fitting a polynomial function to the data, rather than a straight line. It can be used when the relationship between the variables is more complex than a simple linear relationship.
- **Ridge regression:** This type of regression is a regularized form of linear regression that helps prevent overfitting by adding a penalty term to the cost function.
- **Lasso regression:** Like Ridge regression, Lasso regression is also a regularized form of linear regression. However, it uses a different penalty term that encourages the model to select a smaller number of features.
- **ElasticNet regression:** This is a combination of Ridge and Lasso regression. It adds both L1 and L2 regularization penalties to the cost function.

These different types of linear regression models can be used to model relationships between variables in many different situations and are commonly used in machine learning and data science applications. I will cover all these. Lets focus on **Simple Linear Regression**

The California Housing Dataset for Simple Linear Regression

The California Housing Dataset contains information on the median income, housing age, and other features for census tracts in California. The dataset was originally published by Pace, R. Kelley and Ronald Barry in their 1997 paper "Sparse Spatial Autoregressions" and is available in the sklearn.datasets module.

The dataset consists of 20,640 instances, each representing a census tract in California. There are eight features in the dataset, including:

- **MedInc:** Median income in the census tract.
- **HouseAge:** Median age of houses in the census tract.
- **AveRooms:** Average number of rooms per dwelling in the census tract.
- **AveBedrms:** Average number of bedrooms per dwelling in the census tract.
- **Population:** Total number of people living in the census tract.
- **AveOccup:** Average number of people per household in the census tract.
- **Latitude:** Latitude of the center of the census tract.
- **Longitude:** Longitude of the center of the census tract.

The target variable in this dataset is the **median house value** in the census tract, which is a continuous variable. This makes the dataset suitable for regression analysis, particularly simple linear regression, which can be used to model the relationship between **median income** and **median house value**.

Step 1: Import the necessary libraries

In [1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
```

```
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')
```

Step 2: Load the dataset

In [2]:
```
# Load the California Housing Dataset from seaborn

california = fetch_california_housing()

# Convert the data to a pandas dataframe
california_df = pd.DataFrame(data=california.data, columns=california.feature_name

# Add the target variable to the dataframe
california_df['MedHouseVal'] = california.target

# Print the first 5 rows of the dataframe
california_df.head()
```

Out[2]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedI
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	

Step 2: Do Exploratory Data Analysis

Step 2(a): Check Shape of Dataframe

In [3]:
```
# Print the shape of the dataframe
print("Data shape:", california_df.shape)
```

```
Data shape: (20640, 9)
```

Step 2(b): Show Descriptive Statistics of each numerical column

In [4]:
```
california_df.describe().T
```

Out[4]:

	count	mean	std	min	25%	50%	7
MedInc	20640.0	3.870671	1.899822	0.499900	2.563400	3.534800	4.7432
HouseAge	20640.0	28.639486	12.585558	1.000000	18.000000	29.000000	37.0000
AveRooms	20640.0	5.429000	2.474173	0.846154	4.440716	5.229129	6.0523
AveBedrms	20640.0	1.096675	0.473911	0.333333	1.006079	1.048780	1.0995
Population	20640.0	1425.476744	1132.462122	3.000000	787.000000	1166.000000	1725.0000
AveOccup	20640.0	3.070655	10.386050	0.692308	2.429741	2.818116	3.2822
Latitude	20640.0	35.631861	2.135952	32.540000	33.930000	34.260000	37.7100
Longitude	20640.0	-119.569704	2.003532	-124.350000	-121.800000	-118.490000	-118.0100
MedHouseVal	20640.0	2.068558	1.153956	0.149990	1.196000	1.797000	2.6472

Step 2(c): Check for missing values in the Dataframe

In [5]:
```
# Check for missing values
print("Missing values:\n", california_df.isnull().sum())
```

```
Missing values:
 MedInc         0
HouseAge       0
AveRooms       0
AveBedrms      0
Population     0
AveOccup       0
Latitude       0
Longitude      0
MedHouseVal    0
dtype: int64
```

Step 2(d): Create a Correlation Heatmap

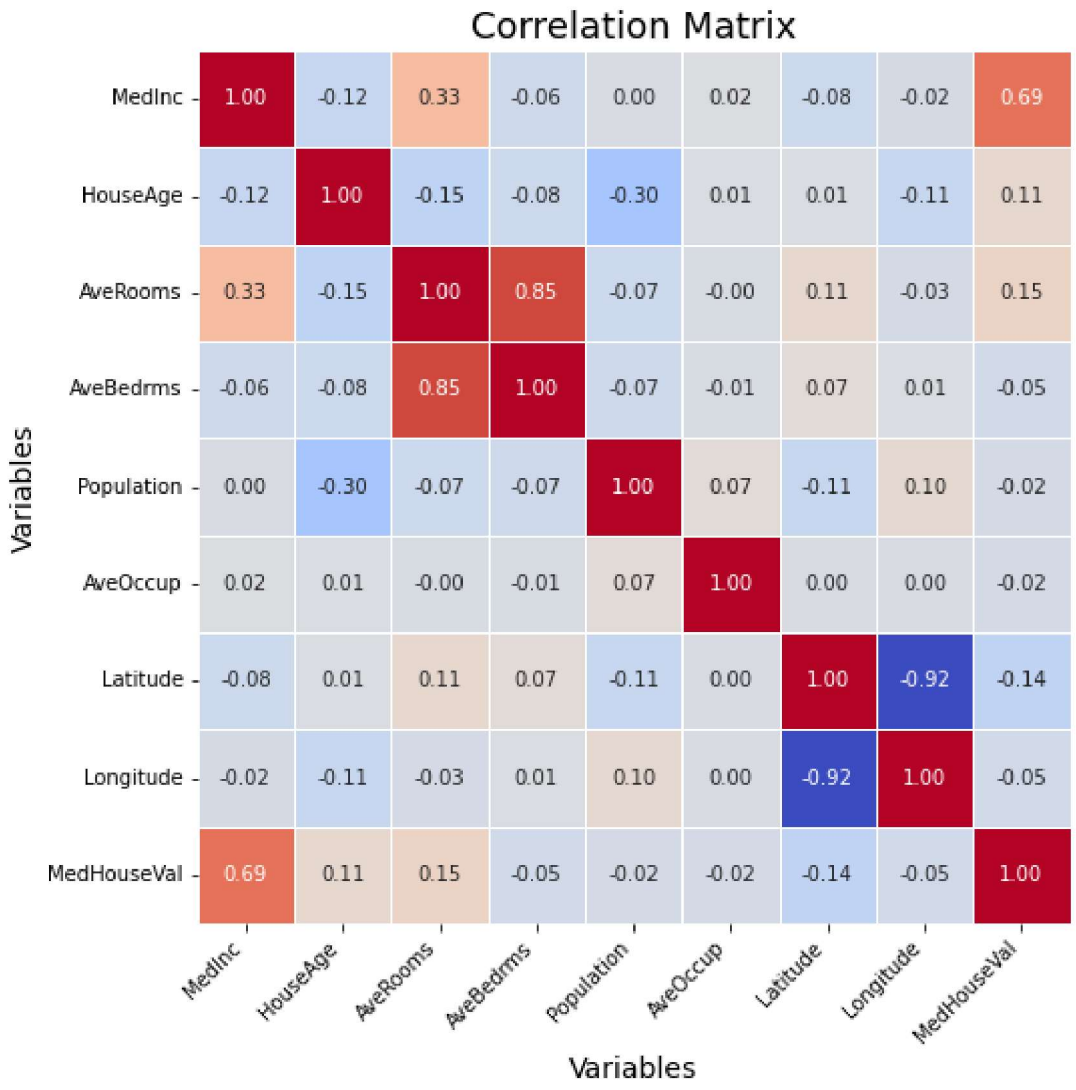
The correlation matrix shows the correlation coefficients between every pair of variables in the dataset. A correlation coefficient ranges from -1 to 1 and measures the strength and direction of the linear relationship between two variables. A coefficient of -1 indicates a perfect negative correlation, a coefficient of 0 indicates no correlation, and a coefficient of 1 indicates a perfect positive correlation.

By analyzing the correlation matrix, we can determine which variables have the strongest correlation with the target variable **(in this case, median income)**. These variables would be the best candidates for the independent variable in a simple linear regression model. We want to choose a variable with a high positive correlation coefficient because this indicates that an increase in that variable is associated with an increase in median income.

In [6]:
```
# Compute the correlation matrix
corr_matrix = california_df.corr()
```

In [7]:
```
# Create a customized heatmap
fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap="coolwarm", linewidths=.5, cb

# Set the plot title and labels
```

```
plt.title("Correlation Matrix", fontsize=18)
plt.xlabel("Variables", fontsize=14)
plt.ylabel("Variables", fontsize=14)

# Customize the tick labels
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right',
ax.set_yticklabels(ax.get_yticklabels(), rotation=0, horizontalalignment='right',

plt.show()
```

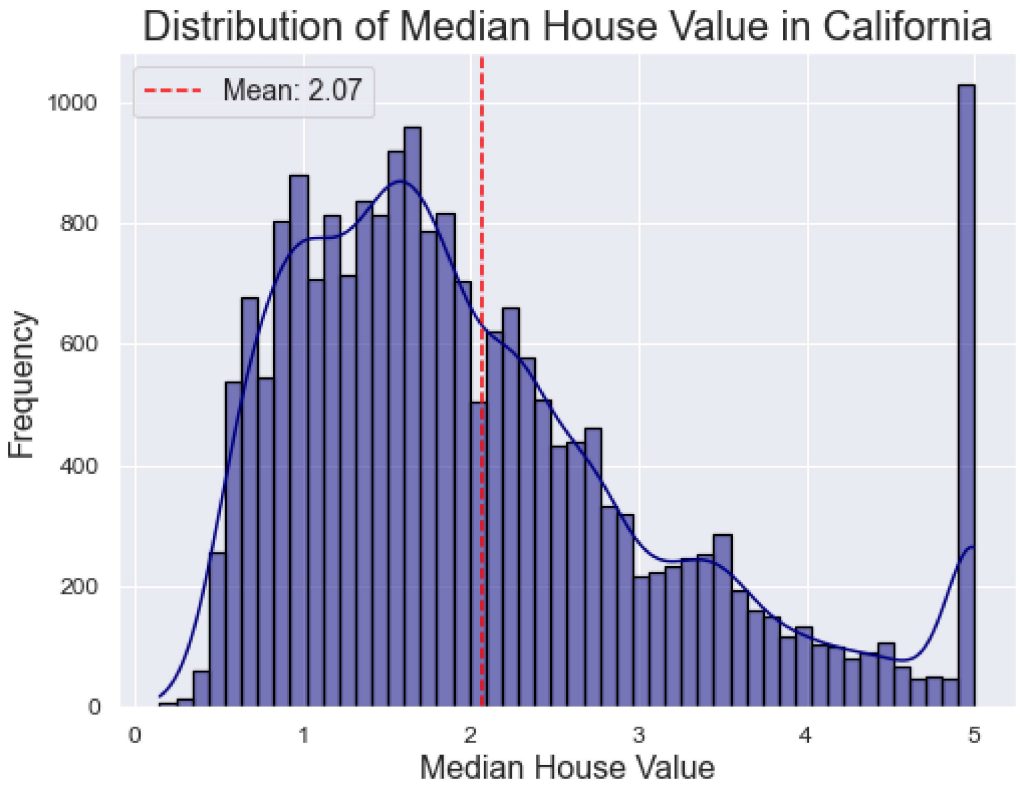


Step 2(e): Visualize the distribution of the target variable using a histogram (Problem: Right Skewed)

A skewed histogram indicates that the data is not normally distributed and has a tendency to cluster towards one end. This can impact the target variable as it may introduce bias in the analysis and modeling process. Additionally, certain statistical measures such as the mean and standard deviation may not be appropriate for skewed data. Therefore, it is important to address skewness in the data before proceeding with any analysis or modeling.

In [8]:
```
# Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='MedHouseVal', kde=True, bins=50, color='navy',

# Set x and y axis labels and title
```

```
plt.xlabel('Median House Value', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of Median House Value in California', fontsize=20)

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for mean
mean = california_df['MedHouseVal'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```

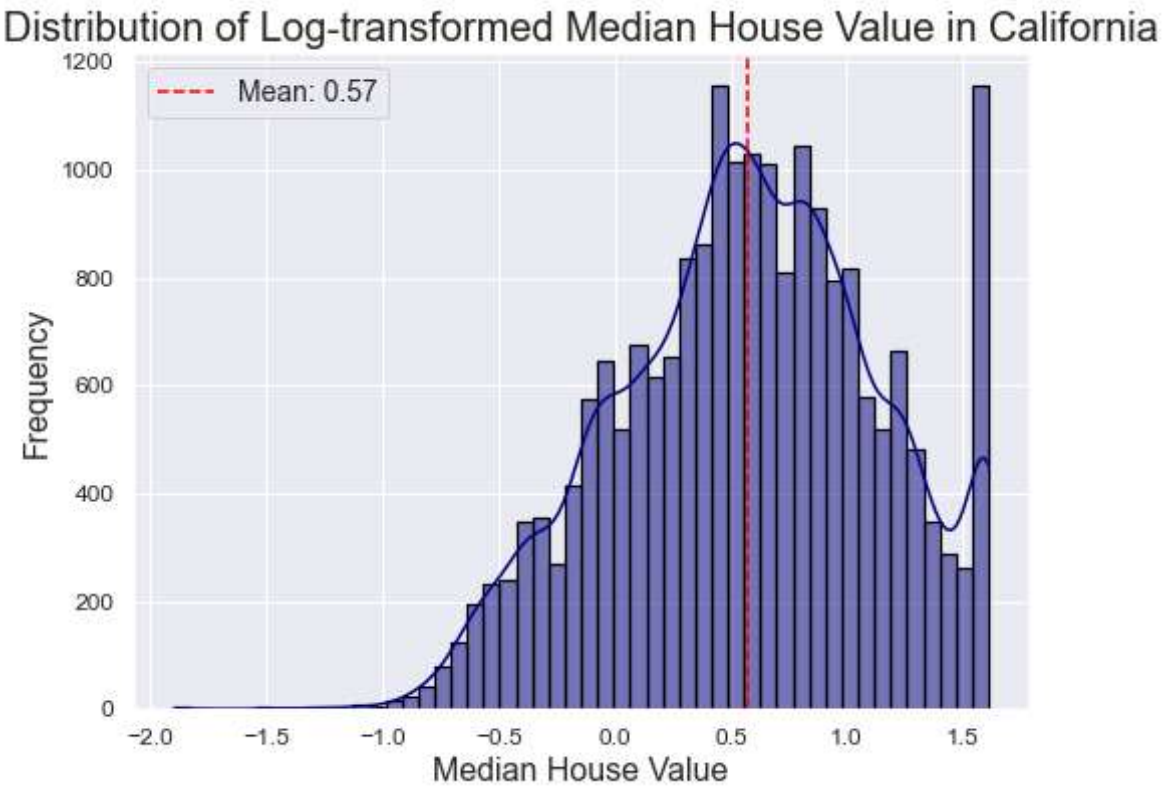


Step 2(e)(i): Right Skewed Resolving: Log-transform the target variable

The histogram of the log-transformed target variable shows a more symmetrical and normal distribution than the original target variable. The log transformation seems to have slightly reduced the right skewness in the distribution of the target variable.

In [9]:
```
# Log-transform the target variable
california_df['MedHouseVal_log'] = np.log(california_df['MedHouseVal'])

# Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='MedHouseVal_log', kde=True, bins=50, color='na

# Set x and y axis labels and title
plt.xlabel('Median House Value', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of Log-transformed Median House Value in California', font
```

```
# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for mean
mean = california_df['MedHouseVal_log'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```

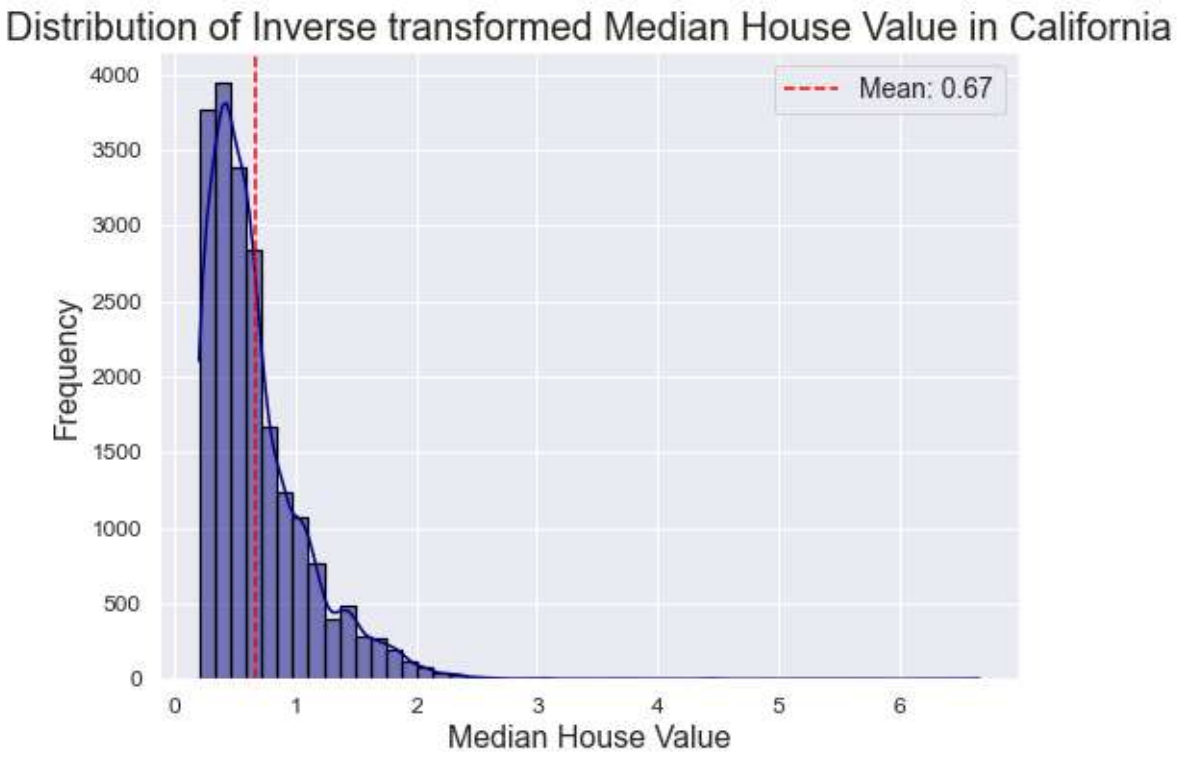


Step 2(e)(ii): Right Skewed Resolving: Inverse Transformation of the target variable

The distribution of the inverse-transformed target variable is left-skewed. The inverse transformation did not improve the distribution of the target variable, and the resulting distribution is not suitable for linear regression analysis.

In [10]:
```
# Apply inverse transformation to target variable
california_df['MedHouseVal_inv'] = 1/california_df['MedHouseVal']

# Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='MedHouseVal_inv', kde=True, bins=50, color='na

# Set x and y axis labels and title
plt.xlabel('Median House Value', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of Inverse transformed Median House Value in California',

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
```

```
# Add vertical line for mean
mean = california_df['MedHouseVal_inv'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```

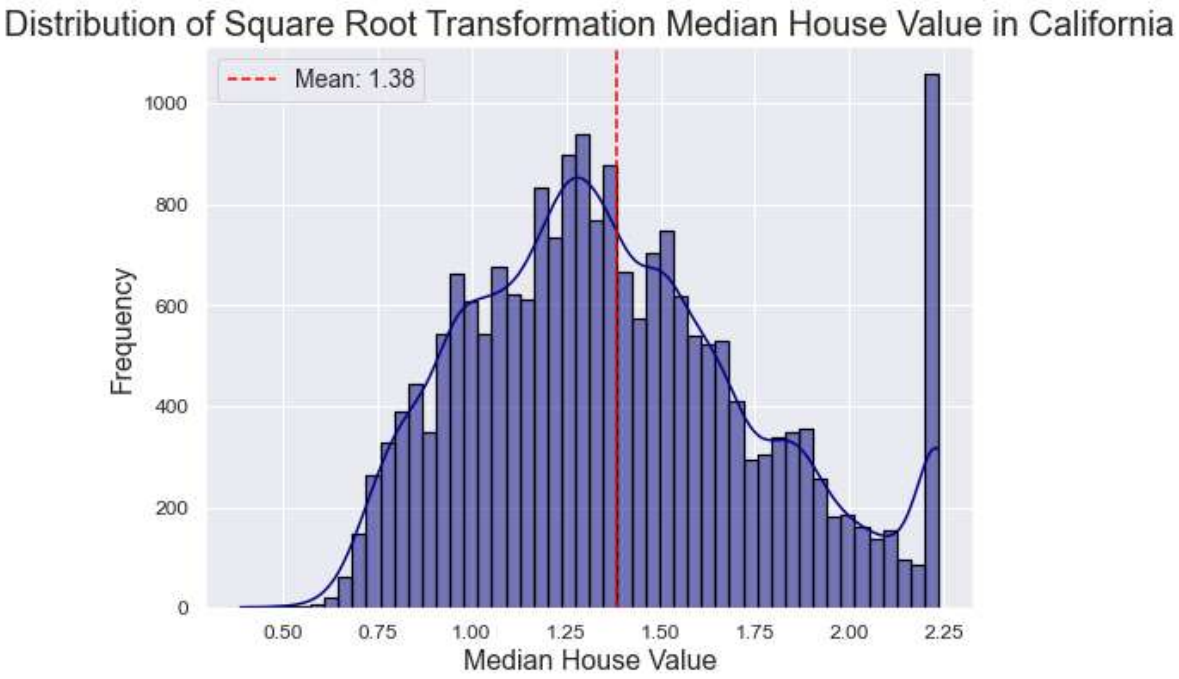


Step 2(e)(iii): Resolve Skew: Square Root Transformation

The resulting histogram shows a more symmetric distribution with less right skewness compared to the original distribution. This indicates that the square root transformation has effectively reduced the skewness in the data.

In [11]:
```
# Apply a square root transformation to the target variable
california_df['sqrt_MedHouseVal'] = np.sqrt(california_df['MedHouseVal'])

# Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='sqrt_MedHouseVal', kde=True, bins=50, color='n

# Set x and y axis labels and title
plt.xlabel('Median House Value', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of Square Root Transformation Median House Value in Califo

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for mean
mean = california_df['sqrt_MedHouseVal'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)
```

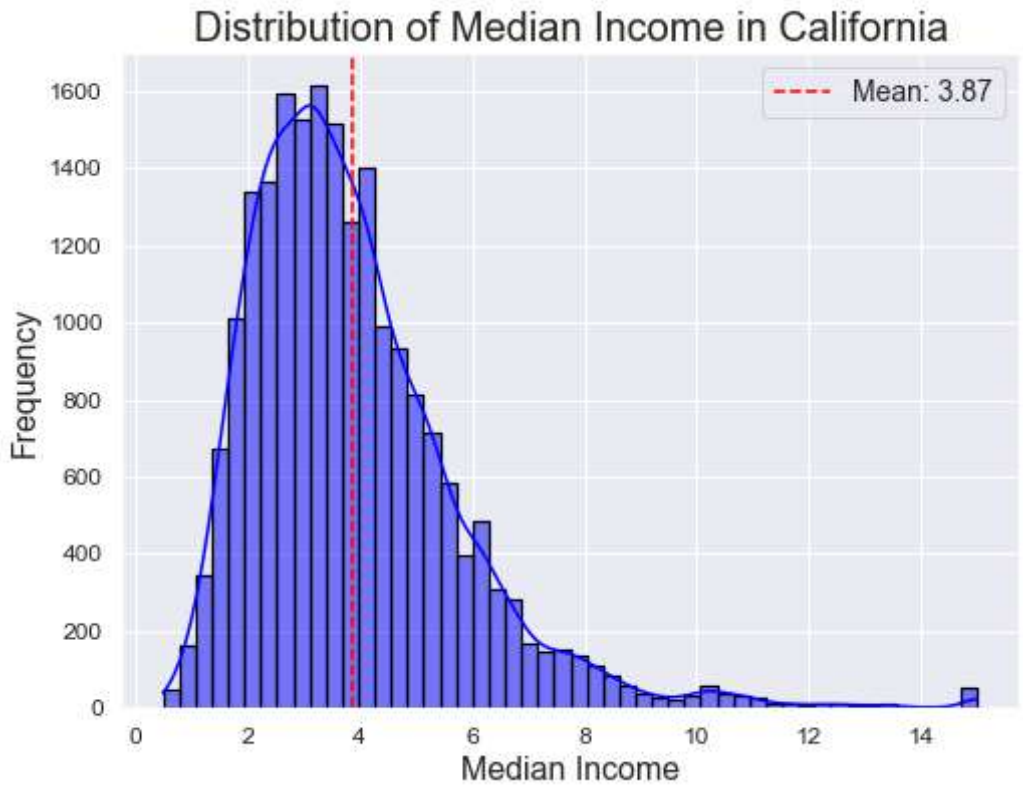
```
# Show the plot
plt.show()
```



Step 2(f): Visualize the distribution of the predictor variable using a histogram (Problem: Outliers)

The histogram shows the distribution of median income in California. The histogram looks normalized however, there appears to be some outliers on the higher end of the distribution. So lets deal with that

In [12]:
```
# Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='MedInc', kde=True, bins=50, color='blue', edge

# Set x and y axis labels and title
plt.xlabel('Median Income', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of Median Income in California', fontsize=20)

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for mean
mean = california_df['MedInc'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```



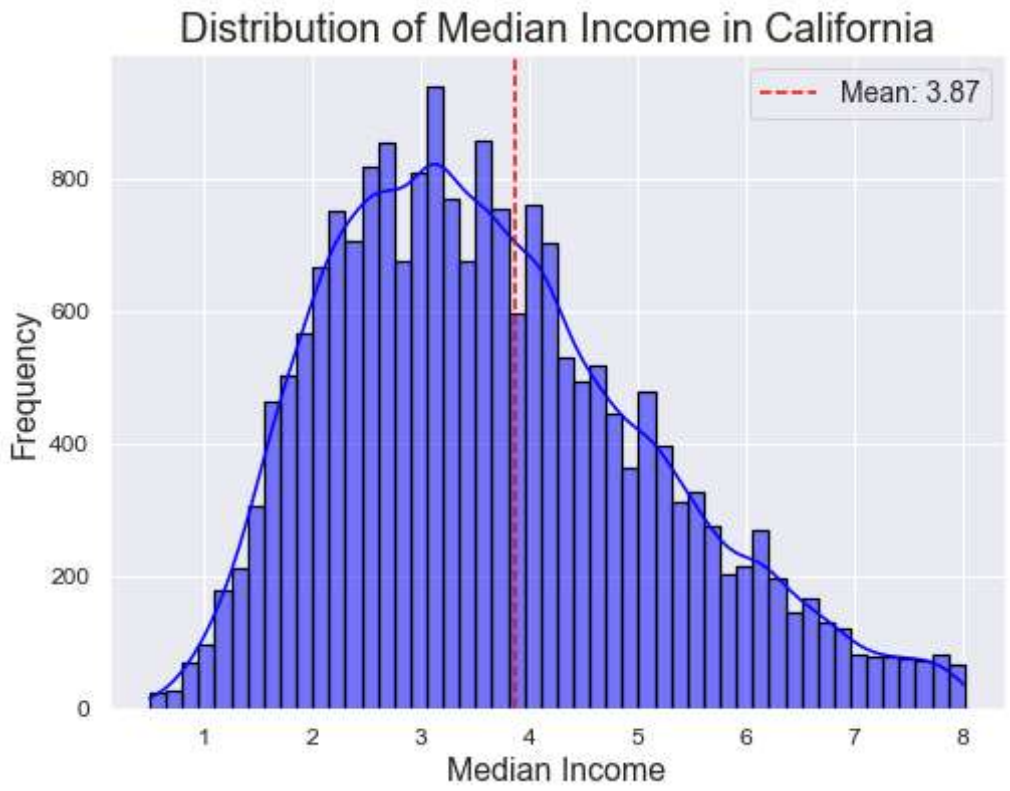
Step 2(f)(i): Resolve Outliers: Using Inter-Quartile Range

First I calculate the first and third quartiles, the interquartile range, and the lower and upper bounds for outlier detection based on the Median Income feature of the California housing dataset. Then I create a new dataframe without outliers by filtering the original dataframe based on whether the Median Income values fall within the calculated bounds. The histogram appears to be more normally distributed, with fewer extreme values compared to the original histogram.

In [13]:
```
# Calculate the first and third quartiles
Q1 = california_df['MedInc'].quantile(0.25)
Q3 = california_df['MedInc'].quantile(0.75)

# Calculate the IQR
IQR = Q3 - Q1

# Define the lower and upper bounds for outlier detection
lower_bound = Q1 - 1.5*IQR
upper_bound = Q3 + 1.5*IQR

# Create a new dataframe without outliers
california_df_no_outliers = california_df[(california_df['MedInc'] >= lower_bound)
```

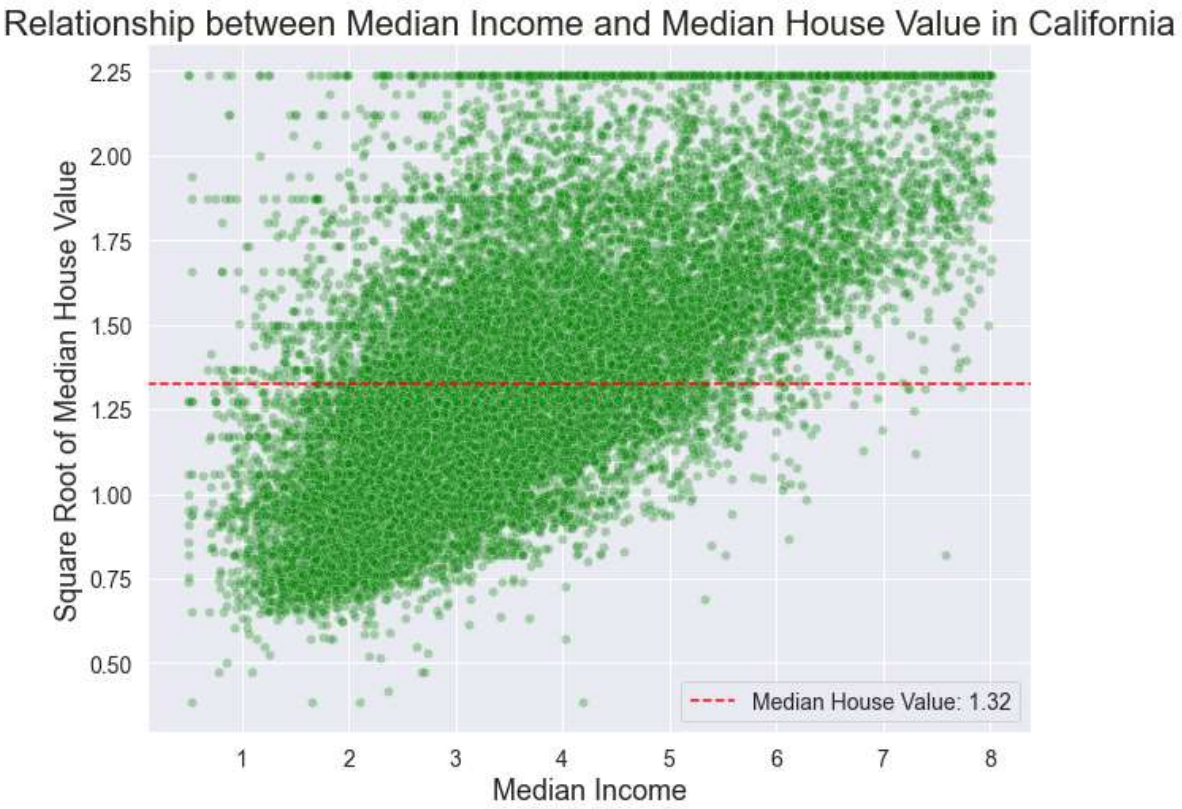
In [14]:
```
# Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df_no_outliers, x='MedInc', kde=True, bins=50, color=

# Set x and y axis labels and title
plt.xlabel('Median Income', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of Median Income in California', fontsize=20)
```

```
# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for mean
mean = california_df['MedInc'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```



In [15]:

```
# Set figure size and font scale
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.5)

# Create scatter plot
sns.scatterplot(data=california_df_no_outliers, x='MedInc', y='sqrt_MedHouseVal',

# Set x and y axis labels and title
plt.xlabel('Median Income', fontsize=18)
plt.ylabel('Square Root of Median House Value', fontsize=18)
plt.title('Relationship between Median Income and Median House Value in California

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

# Add horizontal line for median house value
median_house_value = california_df_no_outliers['sqrt_MedHouseVal'].median()
plt.axhline(median_house_value, color='red', linestyle='--', label=f'Median House
plt.legend(fontsize=14)

# Show the plot
plt.show()
```



Step 3: Do Train-Test Split in the ratio 70-30

In [16]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(california_df_no_outliers[['Me
```

Step 4: Perform Simple Linear Regression

In [17]:
```
# Create an instance of the LinearRegression class
lr = LinearRegression()
```

In [18]:
```
# Create a grid of possible intercept values to try
intercept_grid = {'fit_intercept': [True, False], 'normalize': [True, False], 'copy
```

In [19]:
```
# Use cross-validation to find the best intercept value
grid_search = GridSearchCV(lr, intercept_grid, cv=5)
grid_search.fit(X_train, y_train)
```

Out[19]:
```
GridSearchCV(cv=5, estimator=LinearRegression(),
             param_grid={'copy_X': [True, False],
                         'fit_intercept': [True, False],
                         'normalize': [True, False]})
```

In [20]:
```
# Fit the linear regression model with the best intercept value
lr = grid_search.best_estimator_
lr.fit(X_train, y_train)
```

Out[20]:
```
LinearRegression(normalize=True)
```

In [21]:
```
# Predict the median house value using the test data
y_pred = lr.predict(X_test)
```

Step 4: Evalute the Performance of the Model

The mean squared error (MSE) measures how close the predicted values are to the actual values, with lower values indicating better performance. In this case, the MSE is 0.08, which is relatively low.

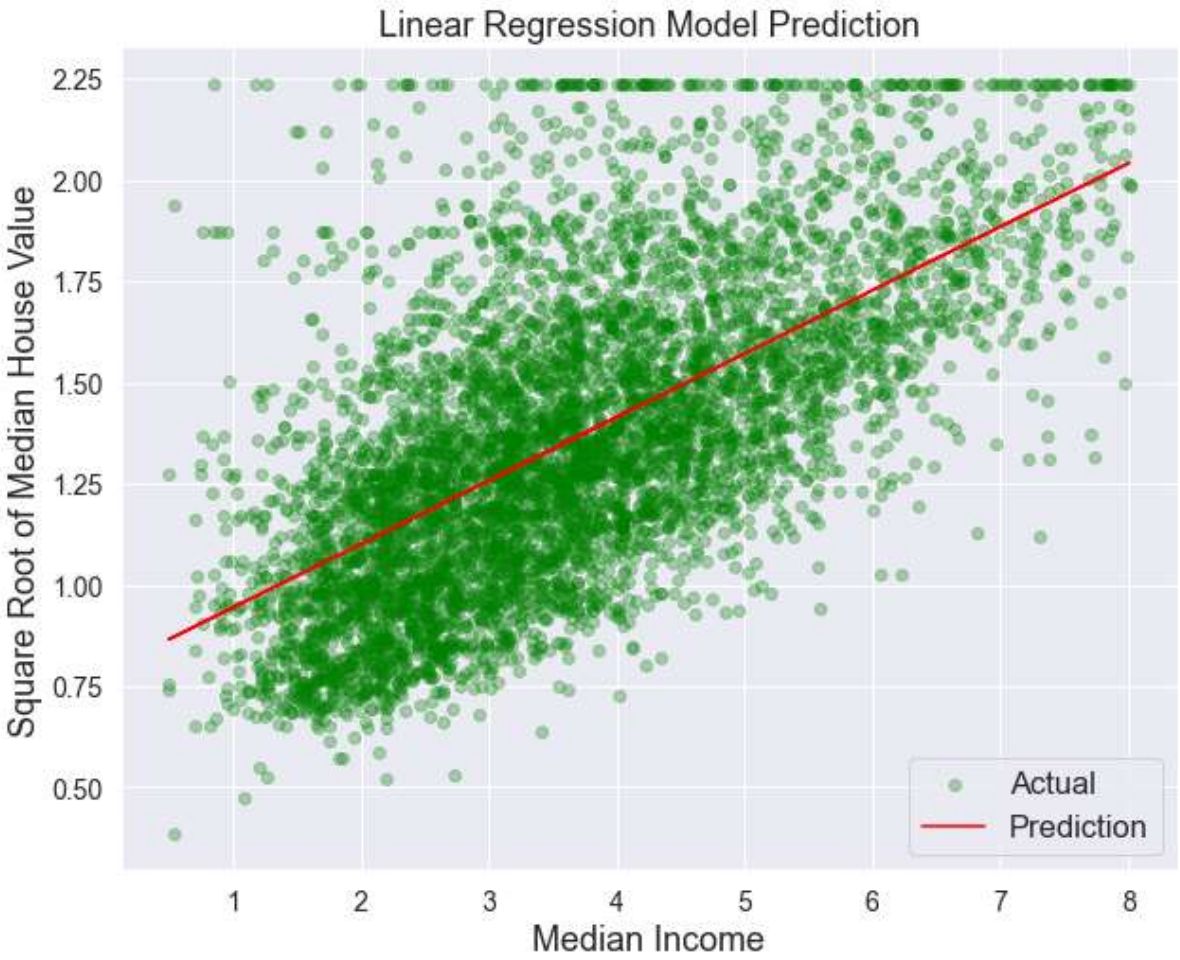
The R-squared score measures how well the regression model fits the actual data, with values closer to 1 indicating a better fit. The R-squared score here is 0.41, which means that only 41% of the variation in median house value can be explained by the median income variable. This indicates that the model has some predictive power, but there is still a lot of variability in the data that is not explained by the model.

In [22]:
```
# Evaluate the model using mean squared error and R-squared score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the mean squared error and R-squared score
print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
```

```
Mean Squared Error: 0.07933878107204766
R-squared Score: 0.4137376961353789
```

Step 4(a): Visualize the Regression

In [23]:
```
# Set figure size and font scale
plt.figure(figsize=(10, 8))

# Plot the linear regression model's prediction on the test set
plt.scatter(X_test, y_test, color='green', label='Actual', alpha=0.3)
plt.plot(X_test, y_pred, color='red', label='Prediction')
plt.xlabel('Median Income')
plt.ylabel('Square Root of Median House Value')
plt.title('Linear Regression Model Prediction')
plt.legend()

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

plt.show()
```



Conclusion

In this simple linear regression tutorial, we explored how to use linear regression to model the relationship between two variables, specifically the median income and median house value in California. We started by loading and cleaning the dataset, and then visualizing the relationship between the two variables.

Next, we split the data into training and testing sets and used linear regression to fit a model to the training data. We then used the model to make predictions on the test data and evaluated the model's performance using mean squared error and R-squared score.

To further optimize our model, we used hyperparameter tuning with GridSearchCV to find the best intercept value. Finally, we fit the linear regression model with the best intercept value, made predictions on the test data, and evaluated the model's performance using mean squared error and R-squared score.

Overall, simple linear regression is a powerful tool for modeling the relationship between two variables and making predictions. However, it is important to properly clean and preprocess the data, and to evaluate the model's performance using appropriate metrics.