In this project, I will build a machine learning model which will predict house price.

Firstly, I will load the data and clean the data and check the data set.

Default title text

```
# @title Default title text
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset from CSV (replace with your actual file path)
df = pd.read_csv("california_housing.csv")

# Check the first few rows
df.head()

```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Next steps: View recommended plots New interactive sheet

```
df.info()
df['ocean_proximity'].unique()
```

```
<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
array(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'],
      dtype=object)
```

```
df.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

```
df.isnull().sum()
# Get the unique values and their frequency in the 'total_bedrooms' column
bedroom_counts = df['total_bedrooms'].value_counts().sort_values

# Display the results
print(bedroom_counts)
```

```
<bound method Series.sort_values of total_bedrooms
280.0     55
331.0     51
345.0     50
343.0     49
393.0     49
          ..
2009.0     1
2249.0     1
```

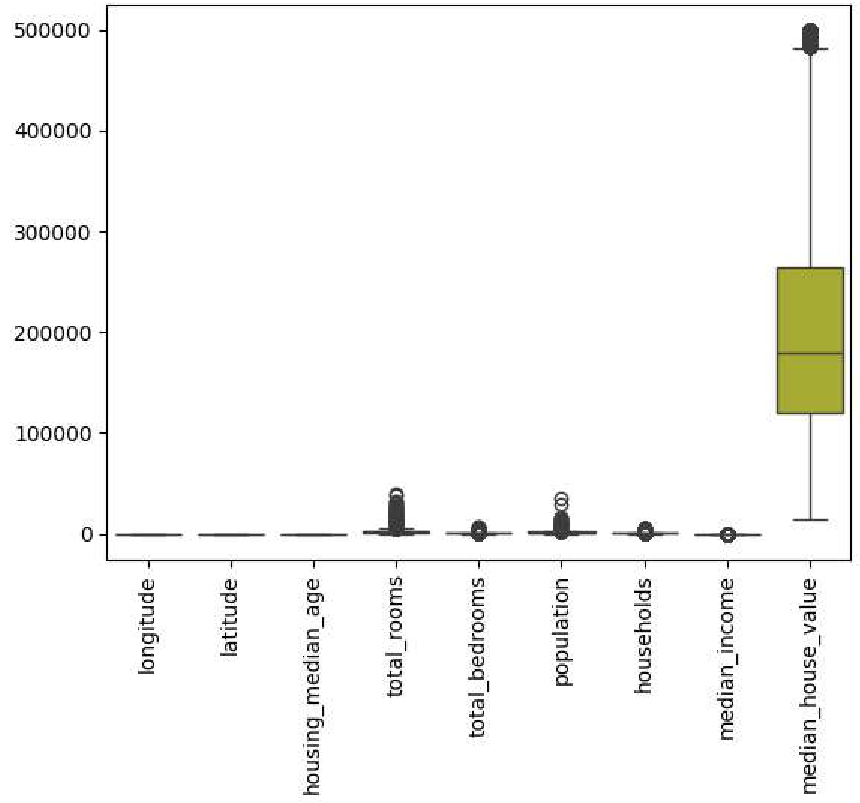
```
3114.0     1
1625.0     1
1742.0     1
Name: count, Length: 1923, dtype: int64>
```

As I assume that a house must have at least 1 bedroom and therefore, I substitute 1 at the places of null values in bedroom column. And as you can see in the box plot, there are no outliers.

```python
# Fill missing values in 'total_bedrooms' column with 1
df['total_bedrooms'] = df['total_bedrooms'].fillna(1)

```

```python
# Visualize data distribution to check for outliers
sns.boxplot(data=df)
plt.xticks(rotation=90)
plt.show()
```



Preparing for XGBoost Machine Learning.

- Binary columns are created for 5 types of variables in ocean_proximity column.
- We split the data into 80% training and 20% test groups. Here I use Standard Scaler to standardize the features in dataset by transforming data to have a mean of 0 and standard deviation of 1

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Split data into features (X) and target (y)
X = df.drop('median_house_value', axis=1)
y = df['median_house_value']

# Identify categorical columns
categorical_columns = X.select_dtypes(include=['object']).columns

# Perform One-Hot Encoding for categorical columns
X_encoded = pd.get_dummies(X, columns=categorical_columns)


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_encoded.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_I
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	False	False	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	False	False	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	False	False	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	False	False	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	False	False	

Next steps: View recommended plots New interactive sheet

Now, I am building a machine learning model with XG Boost. In this model, we are doing regression with 'reg:squarederror' as the loss function and the number of boosting rounds or tree is 1000 and learning 0.05 and the maximum depth of tree is 6 as default hyperparameter values.

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score

# Initialize the XGBoost regressor
model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=1000, learning_rate=0.05, max_depth=6)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

```

```
Mean Squared Error: 2687655514.744204
R-squared: 0.7948994886808447
```

From the result above you can see R-squared value is 0.79 and I use cross validation to check the result furthermore and you can see the overall R squared value is 0.83 which means 83% of the model variation is explained by my model features and 18% remained unexplained and you can say the model is performing well

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X_train, y_train, cv=5, scoring="r2")
print(f"Cross-Validation R² Scores: {scores}")
print(f"Average R² Score: {scores.mean()}")
```

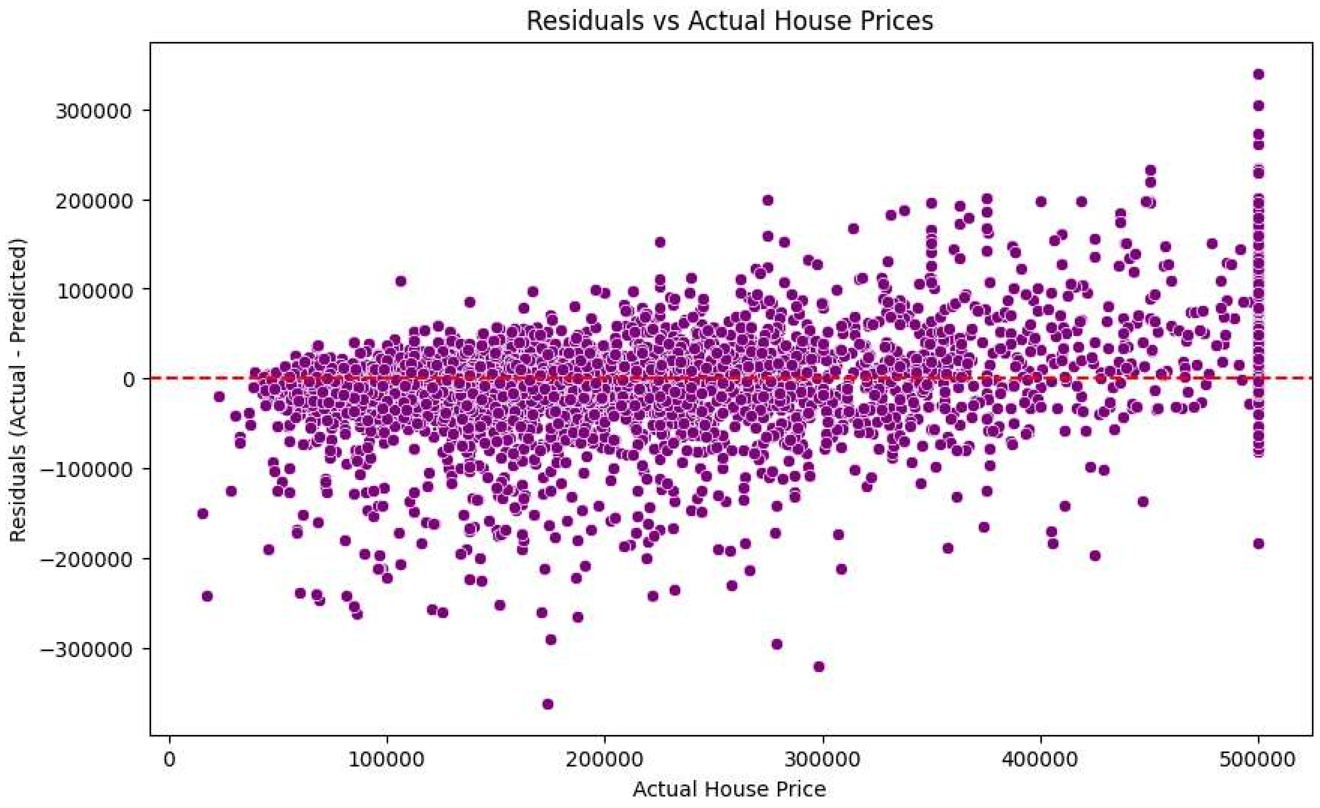
```
Cross-Validation R² Scores: [0.84486807 0.83278214 0.84203853 0.83943947 0.84017939]
Average R² Score: 0.8398615215417877
```

Below is the residual plot, which helps evaluate the model's performance. The residuals (differences between actual and predicted values) are scattered randomly around the red reference line. Since there is no clear pattern in the residual distribution, we can conclude that the model is performing well, indicating that the errors are randomly distributed.

```python
# Calculate the residuals (differences between actual and predicted values)
residuals = y_test - y_pred

# Plot residuals
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=residuals, color='purple')

plt.axhline(0, color='red', linestyle='--')  # Ideal line at y = 0
plt.title('Residuals vs Actual House Prices')
plt.xlabel('Actual House Price')
plt.ylabel('Residuals (Actual - Predicted)')
plt.show()
```



After training the model, I am going to optimize tuning hyper parameters like n_estimators, learning_rate, max_depth using tools like GridSearchCV

```python
from sklearn.model_selection import GridSearchCV

```

```
# Define hyperparameters to tune
param_grid = {
    'n_estimators': [1000, 1200, 1500],
    'learning_rate': [0.1,0.3,0.4,0.5],
    'max_depth': [3, 6, 10]
}

# Perform Grid Search with cross-validation
grid_search = GridSearchCV(xgb.XGBRegressor(objective='reg:squarederror'), param_grid, cv=3, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Best parameters from grid search
print("Best Parameters: ", grid_search.best_params_)

```

```
Best Parameters:  {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 1200}
```

So, I am training a new model with best parameters. {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 1200}

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score

# Initialize the XGBoost regressor
model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=1200, learning_rate=0.1, max_depth=6)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

```

```
Mean Squared Error: 3022138276.9204564
R-squared: 0.7693744222527084
```

Now, I am visulizing the actual and predicted to see how well they match. You can see the dots are even spread throughout the red line and the model is well performing.

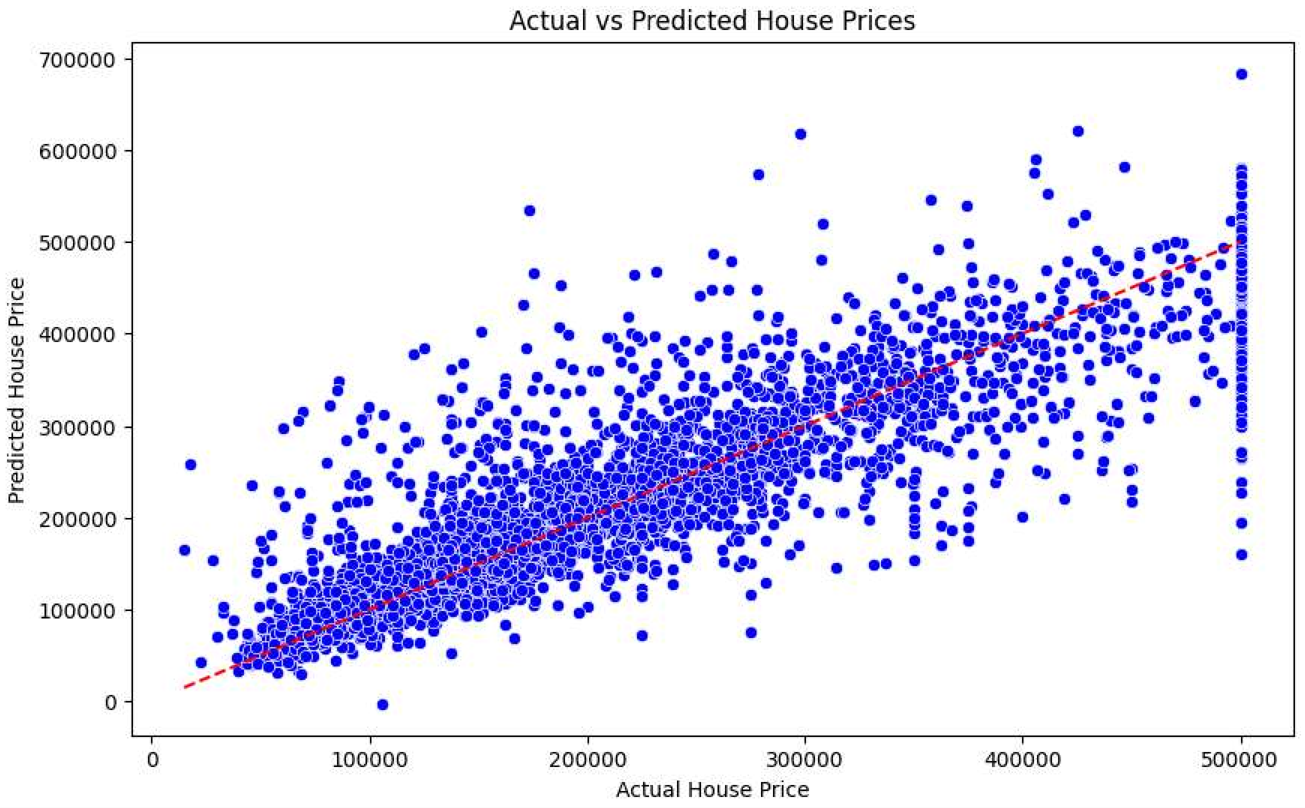
```python
# Create a DataFrame to hold the actual vs predicted values
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Plot the actual vs predicted values
plt.figure(figsize=(10, 6))
sns.scatterplot(data=comparison_df, x='Actual', y='Predicted', color='blue')

# Add a line of perfect prediction (y = x)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')

plt.title('Actual vs Predicted House Prices')
plt.xlabel('Actual House Price')
plt.ylabel('Predicted House Price')
plt.show()


```



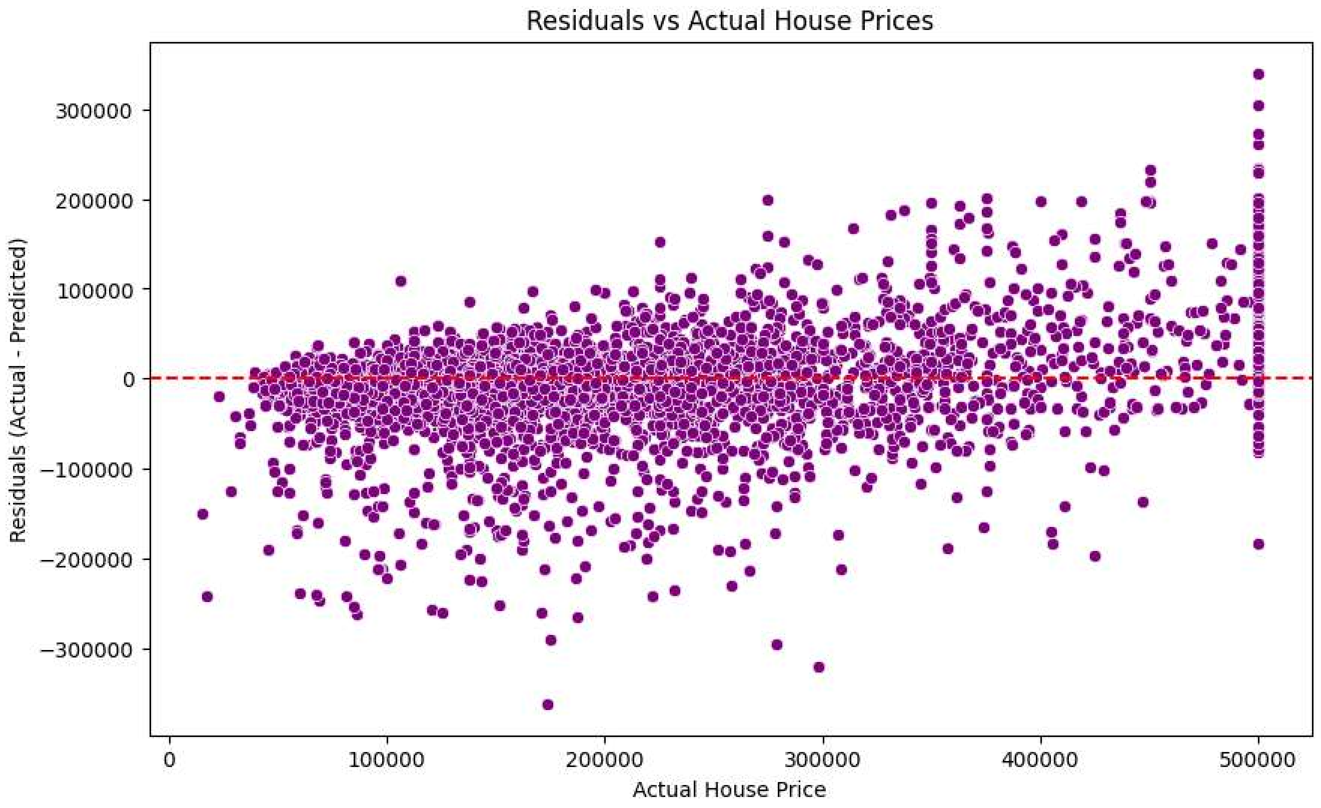
Instead of improving the model's performance, the R-squared value decreased from 0.79 to 0.76, indicating a slight decline in accuracy. However, the residual plot shows that the residuals are more tightly clustered around the red reference line. This suggests that while the model's variance has reduced, it may have lost some predictive power

```
# Calculate the residuals (differences between actual and predicted values)
residuals = y_test - y_pred

# Plot residuals
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=residuals, color='purple')

plt.axhline(0, color='red', linestyle='--')  # Ideal line at y = 0
plt.title('Residuals vs Actual House Prices')
plt.xlabel('Actual House Price')
plt.ylabel('Residuals (Actual - Predicted)')
plt.show()

```



As shown in the results, the cross-validation R-squared mean score has also decreased by approximately 0.0018 compared to the model with default parameters. Since the tuned model does not provide significant improvement, I will proceed with the default parameters

```python
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X_train, y_train, cv=5, scoring="r2")
print(f"Cross-Validation R² Scores: {scores}")
print(f"Average R² Score: {scores.mean()}")
```

```
Cross-Validation R² Scores: [0.84521146 0.83238389 0.84013603 0.83674848 0.83552197]
Average R² Score: 0.838000367965028
```