```
!pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

```
import pandas as pd
```

```
housing_data = pd.read_csv("/content/sample_data/california_housing_train.csv")
```

```
housing_data.head(5)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250	65500.0

```
housing_data.tail(5)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
16995	-124.26	40.58	52.0	2217.0	394.0	907.0	369.0	2.3571	111400.0
16996	-124.27	40.69	36.0	2349.0	528.0	1194.0	465.0	2.5179	79000.0
16997	-124.30	41.84	17.0	2677.0	531.0	1244.0	456.0	3.0313	103600.0
16998	-124.30	41.80	19.0	2672.0	552.0	1298.0	478.0	1.9797	85800.0
16999	-124.35	40.54	52.0	1820.0	300.0	806.0	270.0	3.0147	94600.0

```
# To calculation of descriptive statistics.
housing_data.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_hou
count	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	1700
mean	-119.562108	35.625225	28.589353	2643.664412	539.410824	1429.573941	501.221941	3.883578	20730
std	2.005166	2.137340	12.586937	2179.947071	421.499452	1147.852959	384.520841	1.908157	11598
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	1499
25%	-121.790000	33.930000	18.000000	1462.000000	297.000000	790.000000	282.000000	2.566375	11940
50%	-118.490000	34.250000	29.000000	2127.000000	434.000000	1167.000000	409.000000	3.544600	18040
75%	-118.000000	37.720000	37.000000	3151.250000	648.250000	1721.000000	605.250000	4.767000	26500
max	-114.310000	41.950000	52.000000	37937.000000	6445.000000	35682.000000	6082.000000	15.000100	50000

```
housing_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17000 entries, 0 to 16999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           17000 non-null  float64
 1   latitude            17000 non-null  float64
 2   housing_median_age  17000 non-null  float64
 3   total_rooms         17000 non-null  float64
 4   total_bedrooms      17000 non-null  float64
 5   population          17000 non-null  float64
 6   households          17000 non-null  float64
 7   median_income       17000 non-null  float64
```

```
 8   median_house_value  17000 non-null  float64
dtypes: float64(9)
memory usage: 1.2 MB
```

```
housing_data.shape
```

```
(17000, 9)
```

```
housing_data_shuffled = housing_data.sample(frac=1)
```

```
housing_data_shuffled.head(5)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
5679	-118.18	33.81	27.0	471.0	132.0	315.0	96.0	1.7500	154200.0
8381	-118.47	35.72	18.0	4754.0	1075.0	1366.0	690.0	2.0694	81200.0
6723	-118.29	33.98	46.0	1118.0	300.0	786.0	254.0	1.4042	110000.0
15875	-122.42	37.74	52.0	2713.0	624.0	1370.0	594.0	4.6547	325700.0
8544	-118.50	34.03	44.0	2146.0	394.0	851.0	355.0	6.4800	500001.0

```
housing_data_shuffled.shape
```

```
(17000, 9)
```

```
housing_data_no_duplicates = housing_data_shuffled.drop_duplicates()
```

```
housing_data_no_duplicates.shape
```

```
(17000, 9)
```

```
import numpy as np
```

```
'''import pandas as pd
import numpy as np

# Load the dataset
file_path = '/content/sample_data/california_housing_train.csv'
housing_data_no_duplicates = pd.read_csv(file_path)'''

# Simulate ocean proximity based on longitude
# This is a simplified example; real-world application would require more sophisticated logic
def determine_ocean_proximity(longitude):
    if longitude < -122:
        return 'NEAR OCEAN'
    elif longitude < -121:
        return 'NEAR BAY'
    else:
        return 'INLAND'

# Apply the function to create the ocean_proximity column
housing_data_no_duplicates['ocean_proximity'] = housing_data_no_duplicates['longitude'].apply(determine_ocean_proximity)

# Perform one-hot encoding on the 'ocean_proximity' column
data_encoded = pd.get_dummies(housing_data_no_duplicates, columns=['ocean_proximity'])

# Display the updated DataFrame
data_encoded.head(5)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
5679	-118.18	33.81	27.0	471.0	132.0	315.0	96.0	1.7500	154200.0
8381	-118.47	35.72	18.0	4754.0	1075.0	1366.0	690.0	2.0694	81200.0
6723	-118.29	33.98	46.0	1118.0	300.0	786.0	254.0	1.4042	110000.0
15875	-122.42	37.74	52.0	2713.0	624.0	1370.0	594.0	4.6547	325700.0
8544	-118.50	34.03	44.0	2146.0	394.0	851.0	355.0	6.4800	500001.0

```
# Replace True and False with 1 and 0 in the 'ocean_proximity' columns
# Assuming you have a column named 'ocean_proximity' with boolean values
# Replace 'ocean_proximity' with your actual column name if it's different.

for column in data_encoded.columns:
  if 'ocean_proximity' in column:
    data_encoded[column] = data_encoded[column].astype(int)


# Display the updated DataFrame
data_encoded.head(5)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
5679	-118.18	33.81	27.0	471.0	132.0	315.0	96.0	1.7500	154200.0
8381	-118.47	35.72	18.0	4754.0	1075.0	1366.0	690.0	2.0694	81200.0
6723	-118.29	33.98	46.0	1118.0	300.0	786.0	254.0	1.4042	110000.0
15875	-122.42	37.74	52.0	2713.0	624.0	1370.0	594.0	4.6547	325700.0
8544	-118.50	34.03	44.0	2146.0	394.0	851.0	355.0	6.4800	500001.0

```
# Assuming 'median_house_value' is the column you want to move
cols = list(data_encoded.columns)
cols.remove('median_house_value')
cols.append('median_house_value')
data_encoded = data_encoded[cols]

# Display the updated DataFrame
data_encoded.head(5)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity_INLA
5679	-118.18	33.81	27.0	471.0	132.0	315.0	96.0	1.7500	
8381	-118.47	35.72	18.0	4754.0	1075.0	1366.0	690.0	2.0694	
6723	-118.29	33.98	46.0	1118.0	300.0	786.0	254.0	1.4042	
15875	-122.42	37.74	52.0	2713.0	624.0	1370.0	594.0	4.6547	
8544	-118.50	34.03	44.0	2146.0	394.0	851.0	355.0	6.4800	

```
data_encoded.shape
```

```
(17000, 12)
```

```
data_encoded=data_encoded.dropna()
```

```
data_encoded.shape
```

```
(17000, 12)
```

```
import numpy as np

# Detect outliers using IQR method for 'median_house_value'
Q1 = data_encoded['median_house_value'].quantile(0.25)
Q3 = data_encoded['median_house_value'].quantile(0.75)
IQR = Q3 - Q1
outliers = data_encoded[(data_encoded['median_house_value'] < Q1 - 1.5 * IQR) | (data_encoded['median_house_value'] > Q3 + 1.5 * IQR)]

# Remove outliers
data_no_outliers = data_encoded[(data_encoded['median_house_value'] >= Q1 - 1.5 * IQR) & (data_encoded['median_house_value'] <= Q3 + 1.5 * I
```

```
data_no_outliers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 16105 entries, 16279 to 10026
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   longitude                   16105 non-null  float64
 1   latitude                    16105 non-null  float64
 2   housing_median_age          16105 non-null  float64
 3   total_rooms                 16105 non-null  float64
 4   total_bedrooms              16105 non-null  float64
 5   population                  16105 non-null  float64
 6   households                  16105 non-null  float64
 7   median_income               16105 non-null  float64
 8   ocean_proximity_INLAND      16105 non-null  int64
 9   ocean_proximity_NEAR BAY    16105 non-null  int64
 10  ocean_proximity_NEAR OCEAN  16105 non-null  int64
 11  median_house_value          16105 non-null  float64
dtypes: float64(9), int64(3)
memory usage: 1.6 MB
```

```
data_no_outliers.isnull().sum()
```

	0
longitude	0
latitude	0
housing_median_age	0
total_rooms	0
total_bedrooms	0
population	0
households	0
median_income	0
ocean_proximity_INLAND	0
ocean_proximity_NEAR BAY	0
ocean_proximity_NEAR OCEAN	0
median_house_value	0

```
data_housing=data_no_outliers
```

```
data_housing.shape
```

```
(16105, 12)
```

```
from sklearn.model_selection import train_test_split
```

```
# Separate features and target variable
X = data_housing.drop('median_house_value', axis=1)
y = data_housing['median_house_value']

# First, split the data into training and temporary sets
```

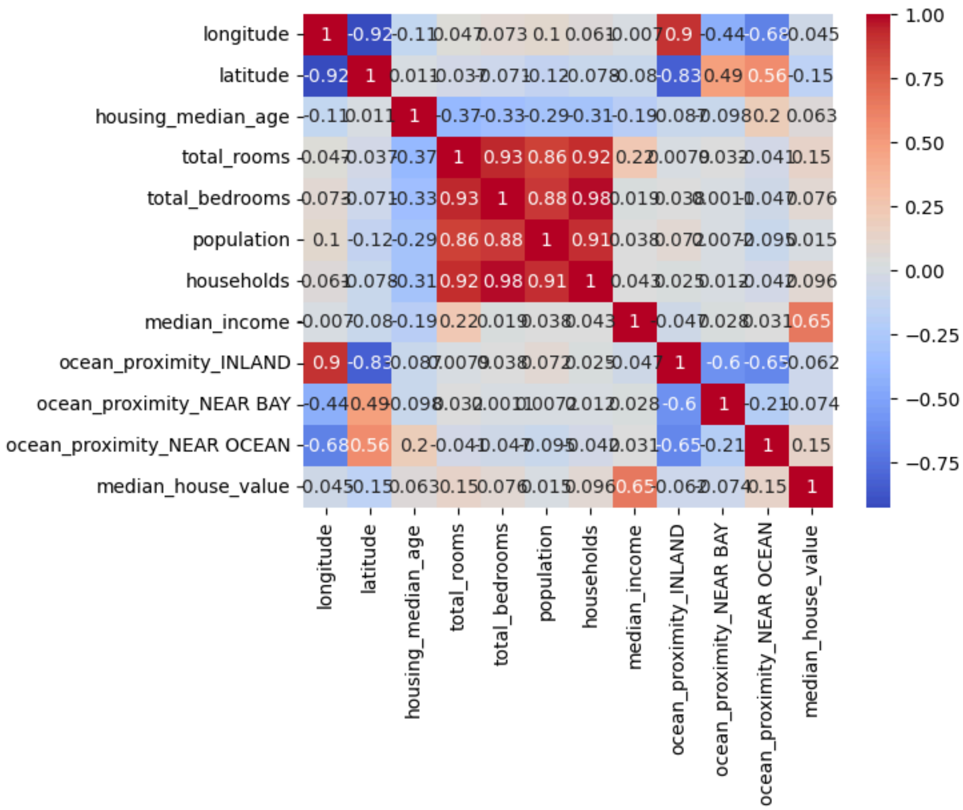
```
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Then, split the temporary set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42)  # 0.25 x 0.8 = 0.2

# Display the sizes of the splits
print(f"Training set size: {X_train.shape,y_train.shape}")
print(f"Validation set size: {X_val.shape,y_val.shape}")
print(f"Test set size: {X_test.shape,y_test.shape}")
```

```
Training set size: ((9663, 11), (9663,))
Validation set size: ((3221, 11), (3221,))
Test set size: ((3221, 11), (3221,))
```

```
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the data
data_scaled = scaler.fit_transform(data_housing.iloc[ : , :8])


# Get the names of the scaled columns
scaled_columns = data_housing.columns[:8]

# Create a DataFrame for the scaled data
data_scaled_df = pd.DataFrame(data_scaled, columns=scaled_columns, index=data_housing.index)

# Concatenate the scaled data with the remaining columns from the original DataFrame
data_processed = pd.concat([data_scaled_df, data_housing.drop(columns=scaled_columns)], axis=1)

# Verify the structure of the processed DataFrame
data_processed.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity_INLA
5679	0.684864	-0.853557	-0.103947	-0.988168	-0.966246	-0.975194	-1.053082	-1.229832	
8381	0.540379	0.033772	-0.824282	0.973252	1.257013	-0.070215	0.482217	-1.025772	
6723	0.630060	-0.774580	1.416761	-0.691871	-0.570161	-0.569632	-0.644703	-1.450759	
15875	-1.427607	0.972203	1.896985	0.038566	0.193716	-0.066771	0.234088	0.625944	
2551	0.948923	-0.700249	-1.544617	0.571626	-0.122209	0.316403	-0.034719	3.072371	

```
import seaborn as sns
import matplotlib.pyplot as plt


# Calculate the correlation matrix
correlation_matrix = data_processed.corr()

# Plot the correlation matrix
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```




```
from sklearn.ensemble import RandomForestRegressor

# Separate features and target variable
X = data_processed.drop('median_house_value', axis=1)
y = data_processed['median_house_value']

# Train a RandomForest model
model = RandomForestRegressor()
model.fit(X, y)

# Get feature importances
feature_importances = model.feature_importances_
features = X.columns

# Display feature importances
importance_df = pd.DataFrame({'Feature': features, 'Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)
print(importance_df)
```

```
                       Feature  Importance
7                median_income    0.452571
0                    longitude    0.205792
1                     latitude    0.180969
2           housing_median_age    0.054208
5                   population    0.032789
3                  total_rooms    0.028335
4               total_bedrooms    0.023475
6                   households    0.020468
9     ocean_proximity_NEAR BAY    0.000733
8       ocean_proximity_INLAND    0.000367
10  ocean_proximity_NEAR OCEAN    0.000294
```

```
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression

# Create and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions and MSE calculation
train_pred = model.predict(X_train)
val_pred = model.predict(X_val)
```

```
train_mse = mean_squared_error(y_train, train_pred)
val_mse = mean_squared_error(y_val, val_pred)

# Print the results
print(f"Training MSE: {train_mse}")
print(f"Validation MSE: {val_mse}")
print(np.sqrt(train_mse))
print(np.sqrt(val_mse))
```

```
Training MSE: 3581875084.283568
Validation MSE: 3600033089.940089
59848.76844416741
60000.2757488671
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

# Initialize the KNN regressor with a chosen K value
k = 9 # You can tune this value
knn = KNeighborsRegressor(n_neighbors=k)

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

# Make predictions on the train and val set
train_pred_knn = knn.predict(X_train)
val_pred_knn = knn.predict(X_val)

# Evaluate the model
ktrain_mse = mean_squared_error(y_train, train_pred_knn)
kval_mse = mean_squared_error(y_val, val_pred_knn)

print(f"KNN Training MSE: {ktrain_mse}")
print(f"KNN Validation MSE: {kval_mse}")
print(np.sqrt(ktrain_mse))
print(np.sqrt(kval_mse))
```

```
KNN Training MSE: 5912027655.560283
KNN Validation MSE: 7700467969.752054
76889.71098632301
87752.31033854354
```

```
from sklearn.ensemble import RandomForestRegressor

# Train a RandomForest model
rfr = RandomForestRegressor(max_depth=10)
rfr.fit(X_train, y_train) # Change: Fit the rfr model, not the 'model' variable

# Make predictions on the train and val set
train_pred_rfr = rfr.predict(X_train)
val_pred_rfr = rfr.predict(X_val)

# Evaluate the model
rtrain_mse = mean_squared_error(y_train, train_pred_rfr)
rval_mse = mean_squared_error(y_val, val_pred_rfr)

print(f"KNN Training MSE: {rtrain_mse}")
print(f"KNN Validation MSE: {rval_mse}")
print(np.sqrt(rtrain_mse))
print(np.sqrt(rval_mse))
```

```
KNN Training MSE: 1261197804.0308385
KNN Validation MSE: 2219313808.976221
35513.346843557825
47109.593598079584
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge

# Define the parameter grid
param_grid = {
    'alpha': [0.1, 1.0, 10.0, 100.0],
    'fit_intercept': [True, False]
}

# Initialize the model
model = Ridge()
```

```
model = Ridge()

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error')

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best Cross-Validation MSE: {-grid_search.best_score_}")
```

```
Best Parameters: {'alpha': 10.0, 'fit_intercept': True}
Best Cross-Validation MSE: 3632873870.138489
```

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error

# Train a Gradient Boosting model with n_estimators
gbr = GradientBoostingRegressor(n_estimators=100)  # Added n_estimators
gbr.fit(X_train, y_train)  # Fit the gbr model

# Make predictions on the train and val set
train_pred_gbr = gbr.predict(X_train)
val_pred_gbr = gbr.predict(X_val)

# Evaluate the model
train_mse_gbr = mean_squared_error(y_train, train_pred_gbr)
val_mse_gbr = mean_squared_error(y_val, val_pred_gbr)

print(f"Gradient Boosting Training MSE: {train_mse_gbr}")
print(f"Gradient Boosting Validation MSE: {val_mse_gbr}")
print(np.sqrt(train_mse_gbr))
print(np.sqrt(val_mse_gbr))
```

```
Gradient Boosting Training MSE: 2170266581.246331
Gradient Boosting Validation MSE: 2443847109.7669673
46586.12004928433
49435.28203385683
```

```
'''import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error


# Define the parameter grids for each model
param_grid_ridge = {
    'alpha': [0.1, 1.0, 10.0, 100.0],
    'fit_intercept': [True, False]
}

param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize the models
ridge = Ridge()
knn = KNeighborsRegressor()
rf = RandomForestRegressor(random_state=42)

# Initialize GridSearchCV for each model
grid_search_ridge = GridSearchCV(estimator=ridge, param_grid=param_grid_ridge, cv=5, scoring='neg_mean_squared_error')
grid_search_knn = GridSearchCV(estimator=knn, param_grid=param_grid_knn, cv=5, scoring='neg_mean_squared_error')
grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
```

```
# Fit GridSearchCV for each model
grid_search_ridge.fit(X_train, y_train)
grid_search_knn.fit(X_train, y_train)
grid_search_rf.fit(X_train, y_train)

# Print the best parameters and best score for each model
print("Linear Regression (Ridge) Best Parameters:", grid_search_ridge.best_params_)
print("Linear Regression (Ridge) Best Cross-Validation MSE:", -grid_search_ridge.best_score_)

print("KNN Best Parameters:", grid_search_knn.best_params_)
print("KNN Best Cross-Validation MSE:", -grid_search_knn.best_score_)

print("Random Forest Best Parameters:", grid_search_rf.best_params_)
print("Random Forest Best Cross-Validation MSE:", -grid_search_rf.best_score_)

# Evaluate the best models on the test set
best_ridge = grid_search_ridge.best_estimator_
best_knn = grid_search_knn.best_estimator_
best_rf = grid_search_rf.best_estimator_

y_pred_ridge = best_ridge.predict(X_test)
y_pred_knn = best_knn.predict(X_test)
y_pred_rf = best_rf.predict(X_test)

mse_ridge = mean_squared_error(y_test, y_pred_ridge)
mse_knn = mean_squared_error(y_test, y_pred_knn)
mse_rf = mean_squared_error(y_test, y_pred_rf)

print(f"Test MSE for Ridge Regression: {mse_ridge}")
print(f"Test MSE for KNN: {mse_knn}")
print(f"Test MSE for Random Forest: {mse_rf}")'''
```