House price prediction

Import the dependencies

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from scipy.stats import uniform, randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
import geopandas as gpd
import folium
from folium.plugins import MarkerCluster
import branca.colormap as cm
from shapely.geometry import Point
```

Upload the data

In [2]:
```
df = pd.read_csv("housing.info.csv")
```

Exploratory Data Analysis

View the first rows of the dataset

In [26]:
```
df.head()
```

Out[26]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342

Understand the shape of the dataset

In [27]:
```
df.shape
```

Out[27]:
```
(20640, 10)
```

Overview about data types, and memory usage of the dataset

In [28]:
```
df.info()
```

```
<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
```

Quick summary about statistics

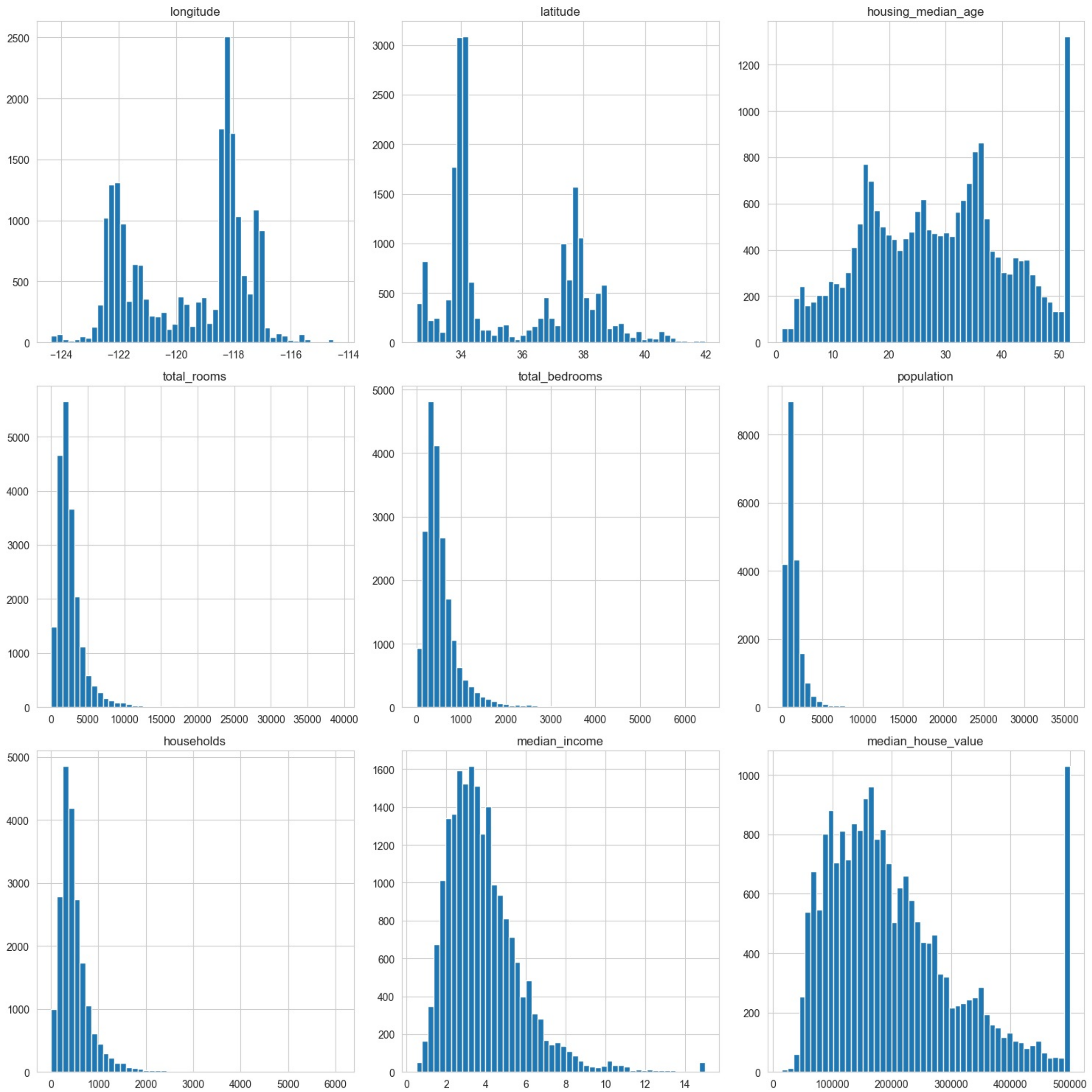
In [29]:
```
df.describe()
```

Out[29]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_incor
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.0000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.8706
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.8998
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.4999
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.5634
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.5348
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.7432
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.0001

Visualize the distributions of the variables

In [30]:
```
df.hist(bins=50, figsize=(15, 15))
plt.tight_layout()
plt.show()
```



- Longitude and latitude present a bimodal distribution with peaks around -122 and -118 for the first one and 34 and 38 for the second one, since in that points are located the major Californian cities like San Francisco (-122, 38) and Los Angeles (-118, 34)

- Housing_median_age looks a bit a uniform distribution, with a spike/peak at 50, probably indicating an age capping/limit
- Total_rooms and total_bedrooms are skewed, most of the values are focused on the lower range, and both have a long tail of large values.
- Population is also skewed with very few large population and many small ones.
- Household present almost an identical distribution to the total_rooms and total_bedrooms, with data skewed and concentrated in the lower range with long tail for large values.
- Median_income is slightly tight-skewed with most values between 2 and 6, tapering off at the higher end.
- Median_house_value has an approximate bell-shaped but it presents a cut-off at 500000, suggesting a cap in the data.

Geospatial Analysis and Visualization

In [ ]:

```python
df['geometry'] = df.apply(lambda row: Point(row['longitude'], row['latitude']), axis=1)
gdf_points = gpd.GeoDataFrame(df, geometry='geometry', crs='EPSG:4326')

# Load California ZIP/Tract polygons
regions = gpd.read_file("ZipCodes_-1049704744535259894.geojson")  # Replace with your file
regions = regions.to_crs('EPSG:4326')

# Spatial Join: assign each point to a region
joined = gpd.sjoin(gdf_points, regions, how='left', predicate='within')

# Aggregate median house value by region
agg = joined.groupby('ZIP_CODE')['median_house_value'].mean().reset_index()
regions = regions.merge(agg, left_on='ZIP_CODE', right_on='ZIP_CODE')

# Create color map
colormap = cm.linear.OrRd_09.scale(regions['median_house_value'].min(), regions['median_house_value'].max())
colormap.caption = 'Median House Value by Region'

# Create map
m6 = folium.Map(location=[36.7783, -119.4179], zoom_start=6, tiles='CartoDB positron')

# Add polygons
folium.GeoJson(
    regions,
    style_function=lambda feature: {
        'fillColor': colormap(feature['properties']['median_house_value']) if feature['properties']['median_hous
        'color': 'black',
        'weight': 0.4,
        'fillOpacity': 0.7
    },
    tooltip=folium.GeoJsonTooltip(fields=['median_house_value'], aliases=['Median House Value: $'])
).add_to(m6)

colormap.add_to(m6)
m6.save("california_choropleth_map.html")
```

Out[ ]:

In the plot is displayed the median house values by ZIP code across California. Darker red ZIP codes present higher median values, they are located around San Francisco, Silicon Valley, Los Angeles and coastal zones. These areas tend to have significantly higher home values due to demand, economic activity and limited housing supply. Beige and light brown have lower house values, they are located mostly inland

Feature Categorization

In [31]:
```
num_features = ['housing_median_age', 'total_rooms', 'total_bedrooms',
                'population', 'households', 'median_income']
cat_features = ['ocean_proximity']
```

Split features and target

In [32]:
```
X = df.drop(columns=['median_house_value'])
y = df['median_house_value']
```

Data Preprocessing Pipeline

In [33]:
```
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OrdinalEncoder(categories=[['INLAND', '<1H OCEAN', 'NEAR OCEAN', 'NEAR BAY', 'ISLAND']]))
])

preprocessor = ColumnTransformer([
    ('num', num_pipeline, num_features),
    ('cat', cat_pipeline, cat_features)
])
```

Model Definition

In [34]:
```
models = {
    'Linear_Regression': LinearRegression(),
    'Ridge': Ridge(alpha=1.0),
    'Lasso': Lasso(alpha=0.1),
    'Random_Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Hist_Gradient_Boosting': HistGradientBoostingRegressor(random_state=42)
}
```

Model Evaluation and Performance Comparison

In [35]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

results = {}

for name, model in models.items():
    pipe = Pipeline([
        ('preprocessing', preprocessor),
        ('regressor', model)
    ])
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)

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

    results[name] = {'RMSE': rmse, 'R2': r2, 'MAE': mae}
    print(f"\n{name}")
    print(f"RMSE: {rmse:.2f}, MAE: {mae:.2f}, R2: {r2:.3f}")
```

```
Linear_Regression
RMSE: 73299.66, MAE: 52810.94, R2: 0.590

Ridge
RMSE: 73297.95, MAE: 52810.50, R2: 0.590

Lasso
RMSE: 73299.62, MAE: 52810.93, R2: 0.590

Random_Forest
RMSE: 63228.04, MAE: 43838.31, R2: 0.695

Hist_Gradient_Boosting
RMSE: 62191.62, MAE: 43198.16, R2: 0.705
```

Histogram-based Graident Boosting is the best model since it has the lowest Root Mean Square Error and Mean Absolute Error and the higher R-Squared that determines the proportion of variance in the dependent variable that can be explained by the independent variable

Hybrid Hyperparameter Tuning

Employ a two-stage hyperparameter tuning process, which is a smart strategy for efficiently optimizing a Histogram-based Gradient Boosting Regressor.

In the first stage we implement Random Search that explores a wide range of hyperparameters quickly. Since exhaustive grid search can be computationally expensive, Random Search samples different hyperparameter combinations efficiently.

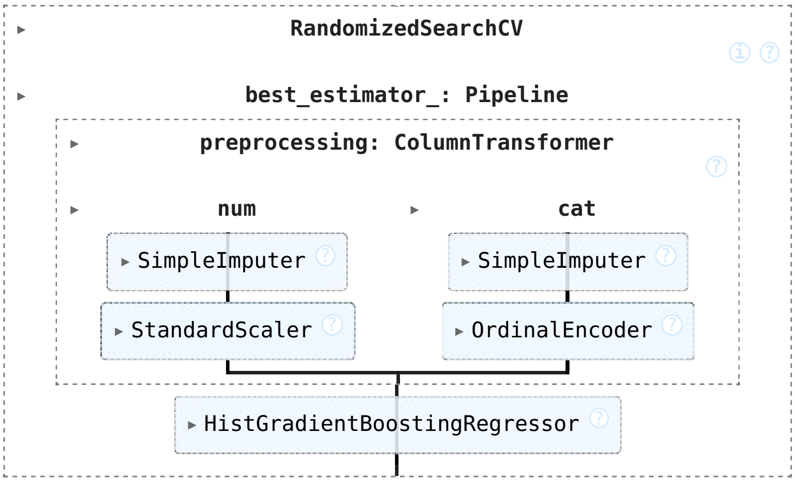
In the Second Stage we apply Grid Search that refines the best parameters by searching in a narrower range around the best values obtained from Random Search. Grid Search exhaustively evaluates all possible combinations in the refined space to determine an even more optimal configuration.

Step 1: Random Search

In [36]:
```
param_dist = {
    'regressor__learning_rate': uniform(0.01, 0.3),          # controls learning step
    'regressor__max_iter': randint(100, 500),                 # number of boosting iterations
    'regressor__max_leaf_nodes': randint(20, 150),            # max leaves in each tree
    'regressor__min_samples_leaf': randint(5, 50),            # minimum samples in a leaf
    'regressor__l2_regularization': uniform(0.0, 1.0),       # L2 regularization to prevent overfitting
    'regressor__max_bins': randint(128, 256)                  # number of bins for histograms
}

random_search = RandomizedSearchCV(
    pipe, param_distributions=param_dist,
    n_iter=20, cv=5, scoring='neg_mean_squared_error', n_jobs=-1, random_state=42
)
random_search.fit(X_train, y_train)
```

Out[36]:



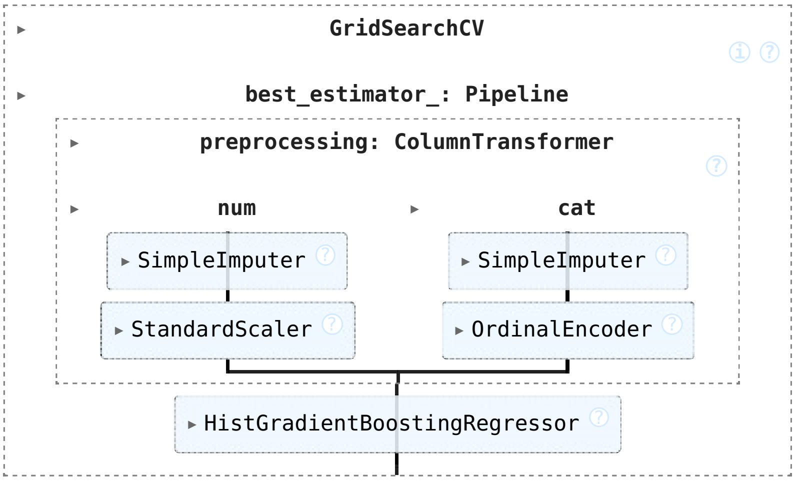
Step 2: Grid Search

In [37]:
```
best_params = random_search.best_params_
refined_grid = {
    'regressor__learning_rate': [round(best_params['regressor__learning_rate'], 2), round(best_params['regresso
    'regressor__max_iter': [best_params['regressor__max_iter'] - 20, best_params['regressor__max_iter'], best_pa
    'regressor__max_leaf_nodes': [best_params['regressor__max_leaf_nodes'] - 10, best_params['regressor__max_lea
}

grid_search = GridSearchCV(
    pipe, param_grid=refined_grid, cv=5,
    scoring='neg_mean_squared_error', n_jobs=-1
)
```

```
grid_search.fit(X_train, y_train)
```

Out[37]:



Best model evaluation

In [38]:

```
y_pred = grid_search.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print("\nBest Parameters after Hybrid Search:", grid_search.best_params_)
print(f"\nOptimized HistGradientBoosting")
print(f"RMSE: {rmse:.2f}, MAE: {mae:.2f}, R2: {r2:.3f}")
```

```
Best Parameters after Hybrid Search: {'regressor__learning_rate': np.float64(0.02), 'regressor__max_iter': 337,
'regressor__max_leaf_nodes': 63}

Optimized HistGradientBoosting
RMSE: 61697.69, MAE: 42804.60, R2: 0.710
```