Import libraries and set up the environment

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import  train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

read the csv file

In [2]:
```
df = pd.read_csv(r"./data/California_Houses.csv")
df
```

Out[2]:

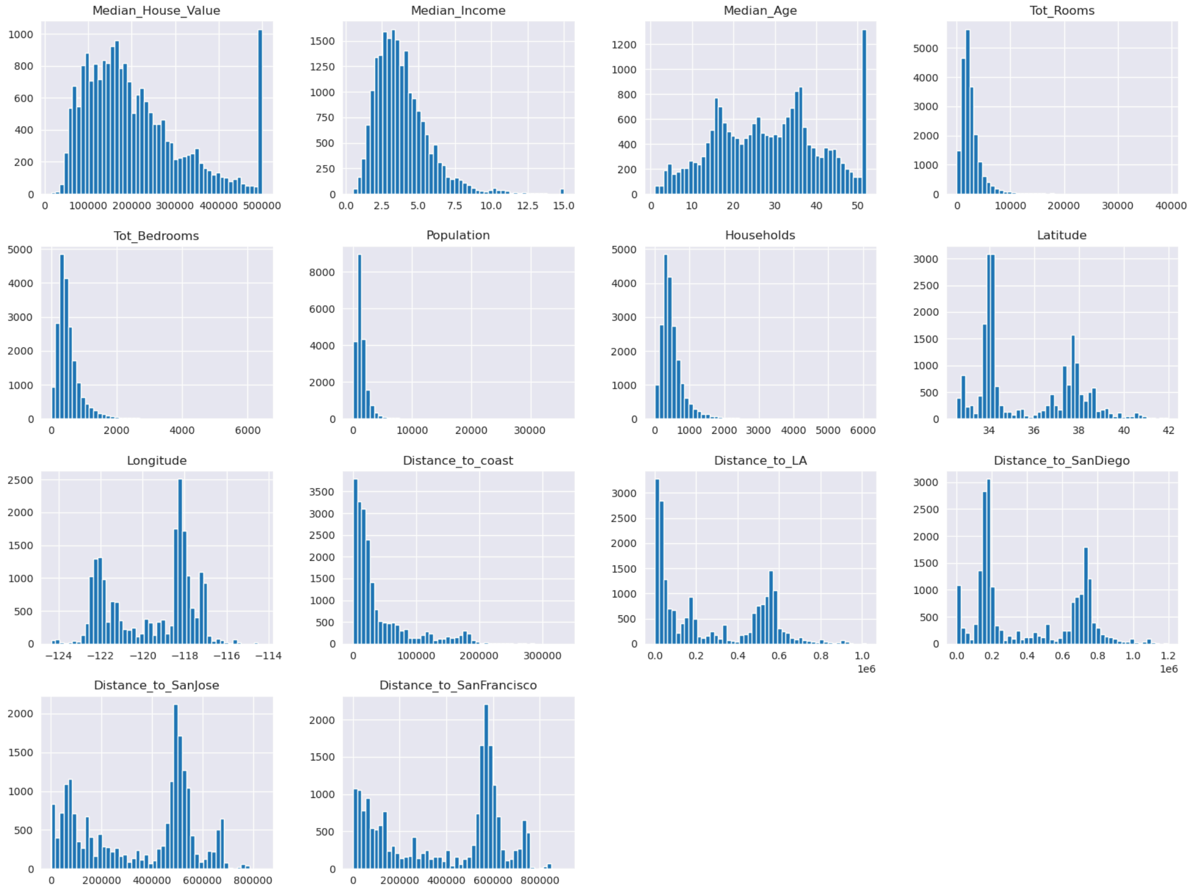
	Median_House_Value	Median_Income	Median_Age	Tot_Rooms	Tot_Bedrooms	Populatio
0	452600	8.3252	41	880	129	32
1	358500	8.3014	21	7099	1106	240
2	352100	7.2574	52	1467	190	49
3	341300	5.6431	52	1274	235	55
4	342200	3.8462	52	1627	280	56
...	...	...	...	...	...	.
20635	78100	1.5603	25	1665	374	84
20636	77100	2.5568	18	697	150	35
20637	92300	1.7000	17	2254	485	100
20638	84700	1.8672	18	1860	409	74
20639	89400	2.3886	16	2785	616	138

20640 rows × 14 columns

Data exploration

Histogram for each attribute

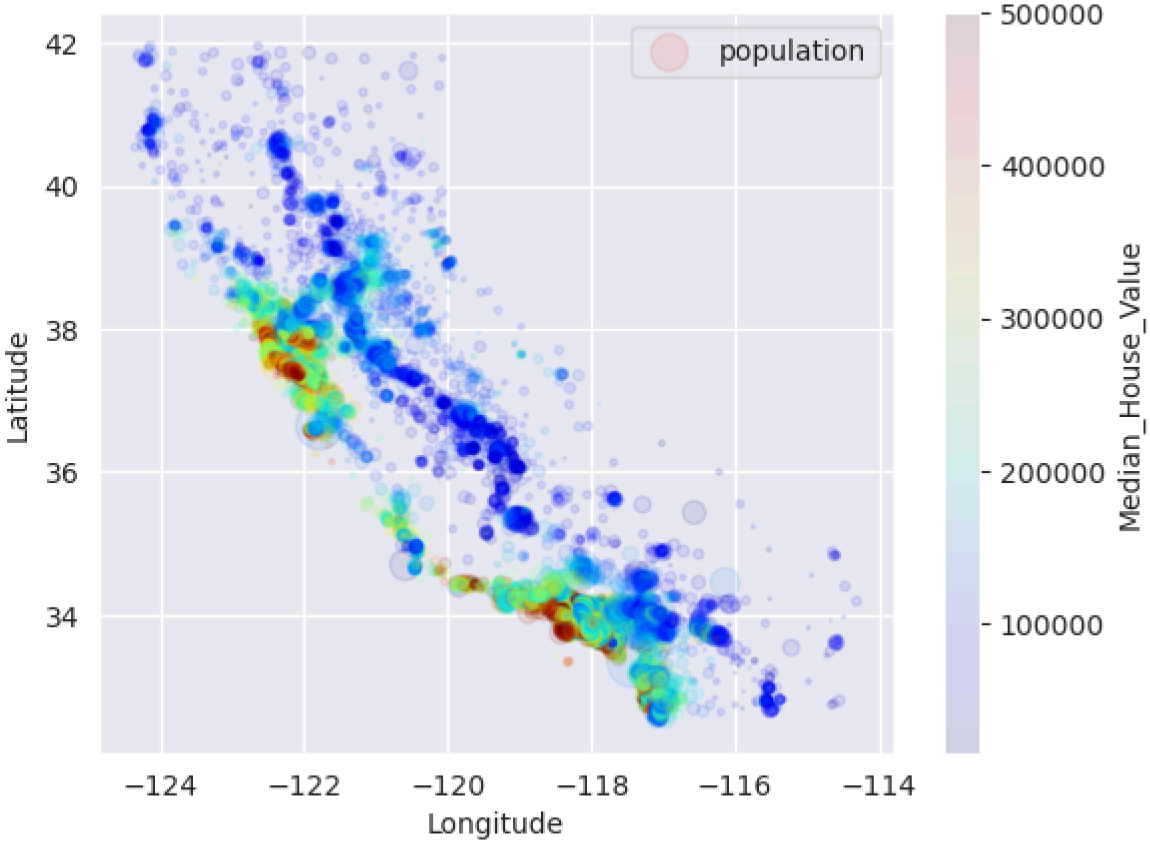
In [3]:
```
df.hist(bins=50, figsize=(20, 15))
plt.show()
```



California map

- using the Latitude and Longitude, we are going to visualize the map of California and see the prices in different areas

In [4]:
```
df.plot(kind="scatter", x="Longitude", y="Latitude", alpha=0.1,
        s=df['Population']/100, label="population",
        c="Median_House_Value", cmap=plt.get_cmap("jet"))
plt.show()
```



- we can see that as we approach the ocean side of California which is at the south-eastern side of the map (around 34 Latitude and -122 Longitude) the prices are relatively higher, especially in popular places like San Francisco and Los Angeles
- the population per district gets smaller around 41 Latitude and -122 Longitude
- population pre district also gets smaller around 36 Latitude and -121 Longitude
- As we go in land (far from the ocean) the prices gets lower

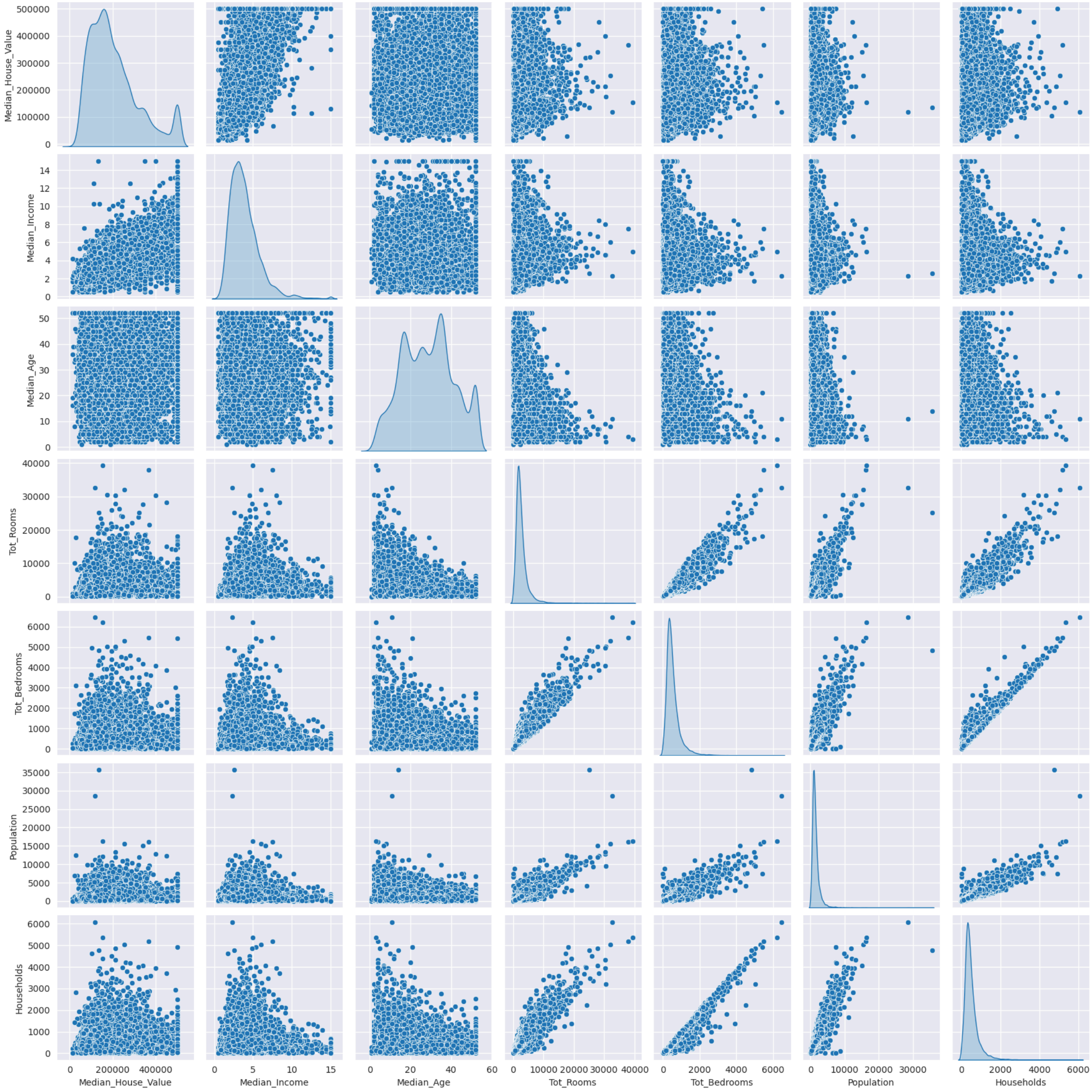
what about correlation ?

In [5]:
```
corr_matrix =df.corr()
corr_matrix["Median_House_Value"].sort_values(ascending=False)
```

Out[5]:
```
Median_House_Value          1.000000
Median_Income               0.688075
Tot_Rooms                   0.134153
Median_Age                  0.105623
Households                  0.065843
Tot_Bedrooms                0.050594
Population                 -0.024650
Distance_to_SanFrancisco   -0.030559
Distance_to_SanJose        -0.041590
Longitude                  -0.045967
Distance_to_SanDiego       -0.092510
Distance_to_LA             -0.130678
Latitude                   -0.144160
Distance_to_coast          -0.469350
Name: Median_House_Value, dtype: float64
```

In [9]:
```
removed_cols = ['Distance_to_LA', 'Distance_to_SanJose', 'Distance_to_SanDie
sns.pairplot(data=df.drop(removed_cols, axis = 1, inplace=False), diag_kind=
plt.show()
```

```
/home/hp/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Data scaling

In [33]:

```python
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)
scaled_data = pd.DataFrame(scaled_data, columns=df.columns)
scaled_data.head()
```

Out[33]:

	Median_House_Value	Median_Income	Median_Age	Tot_Rooms	Tot_Bedrooms	Population	H
0	2.129631	2.344766	0.982143	-0.804819	-0.970706	-0.974429	
1	1.314156	2.332238	-0.607019	2.045890	1.348649	0.861439	
2	1.258693	1.782699	1.856182	-0.535746	-0.825895	-0.820777	
3	1.165100	0.932968	1.856182	-0.624215	-0.719067	-0.766028	
4	1.172900	-0.012881	1.856182	-0.462404	-0.612239	-0.759847	

Data splitting

In [34]:
```
x = scaled_data.drop(['Median_House_Value'], axis=1)
y = scaled_data['Median_House_Value']
x
```

Out[34]:

	Median_Income	Median_Age	Tot_Rooms	Tot_Bedrooms	Population	Households	Latitu
0	2.344766	0.982143	-0.804819	-0.970706	-0.974429	-0.977033	1.0525
1	2.332238	-0.607019	2.045890	1.348649	0.861439	1.669961	1.0431
2	1.782699	1.856182	-0.535746	-0.825895	-0.820777	-0.843637	1.0385
3	0.932968	1.856182	-0.624215	-0.719067	-0.766028	-0.733781	1.0385
4	-0.012881	1.856182	-0.462404	-0.612239	-0.759847	-0.629157	1.0385
...	...	...	...	...	...	...	
20635	-1.216128	-0.289187	-0.444985	-0.389087	-0.512592	-0.443449	1.8016
20636	-0.691593	-0.845393	-0.888704	-0.920853	-0.944405	-1.008420	1.8063
20637	-1.142593	-0.924851	-0.174995	-0.125578	-0.369537	-0.174042	1.7782
20638	-1.054583	-0.845393	-0.355600	-0.305998	-0.604429	-0.393753	1.7782
20639	-0.780129	-1.004309	0.068408	0.185411	-0.033977	0.079672	1.7501

20640 rows × 13 columns

In [35]:
```
#split the data 70:30
x_train, x_validationAndTest, y_train, y_validationAndTest = train_test_spli

#split the 30 50:50
x_validation, x_test, y_validation, y_test = train_test_split(x_validationAn
```

Model interpretation

- model score: the higher, the better
- MSE: the closer to zero the more accurate the prediction is
- MAE: same as MSE, closer to zero means more accurate model

---

helpful resources

- here
- here
- linear regression docs

---

linear regression

In [36]:
```
LR = LinearRegression()
model = LR.fit(x_train, y_train)
```

```
linear_prediction = model.predict(x_validation)

print(f"score: {LR.score(x_validation, y_validation)}")
print(f"MSE: {metrics.mean_squared_error(linear_prediction, y_validation)}")
print(f"MAE: {metrics.mean_absolute_error(y_validation, linear_prediction)}")
```

```
score: 0.6373618777908012
MSE: 0.35680430265844376
MAE: 0.4304488691848636
```

lasso regression

In [37]:
```
lasso = Lasso(max_iter=500)
lasso.fit(x_train, y_train)

lasso_prediction = lasso.predict(x_validation)
print(f"score: {lasso.score(x_validation, y_validation)}")
print(f"MSE: {metrics.mean_squared_error(y_validation, lasso_prediction)}")
print(f"MAE: {metrics.mean_absolute_error(y_validation, lasso_prediction)}")
```

```
score: -0.0009510978129561032
MSE: 0.9848486316734414
MAE: 0.7856244185475799
```

Ridge regression

In [38]:
```
ridge = Ridge()
ridge.fit(x_train, y_train)

ridge_prediction = ridge.predict(x_validation)
print(f"score: {ridge.score(x_validation, y_validation)}")
print(f"MSE: {metrics.mean_squared_error(y_validation, ridge_prediction)}")
print(f"MAE: {metrics.mean_absolute_error(y_validation, ridge_prediction)}")
```

```
score: 0.6374046889133742
MSE: 0.35676218024549217
MAE: 0.4304494187912539
```

Report

In [39]:
```
linear_pred = model.predict(x_test)
lasso_pred = lasso.predict(x_test)
ridge_pred = ridge.predict(x_test)

print('--------------------------\nlinear regression report: \n ------------
print(f"Score: {LR.score(x_test, y_test)}")
print(f"MSE: {metrics.mean_squared_error(y_test, linear_pred)}")
print(f"MAE: {metrics.mean_absolute_error(y_test, linear_pred)}")

print('--------------------------\nLasso regression report: \n -------------
print(f"Score: {lasso.score(x_test, y_test)}")
print(f"MSE: {metrics.mean_squared_error(y_test, lasso_pred)}")
print(f"MAE: {metrics.mean_absolute_error(y_test, ridge_pred)}")

print('--------------------------\nRidge regression report: \n -------------
print(f"Score: {ridge.score(x_test, y_test)}")
print(f"MSE: {metrics.mean_squared_error(y_test, ridge_pred)}")
print(f"MAE: {metrics.mean_absolute_error(y_test, ridge_pred)}")
```

```
-------------------------
linear regression report:
 --------------------------
Score: 0.6481841516436676
MSE: 0.33872284726682705
MAE: 0.42088013544191016
-------------------------
Lasso regression report:
 --------------------------
Score: -0.0009393322383295377
MSE: 0.963688879114195
MAE: 0.42092060594815556
-------------------------
Ridge regression report:
 --------------------------
Score: 0.6481157358464604
MSE: 0.3387887169362474
MAE: 0.42092060594815556
```

Conclusion

- Lasso regression has a very low accuracy
- We can conclude that linear and ridge regressions are more performant than lasso