The purpose of the project is to predict median house values in Californian districts, given many features from these districts.

Our dataset consists of the following features:

1. longitude: A measure of how far west a house is, a higher value is farther west
2. latitude: A measure of how far north a house is, a higher value is farther north
3. housingMedianAge: Median age of a house within a block, a lower number is a newer building
4. totalRooms: Total number of rooms within a block
5. totalBedrooms: Total number of bedrooms within a block
6. population: Total number of people residing within a block
7. households: Total number of households, a group of people residing within a home unit, for a block
8. medianIncome: Median income for households within a block of houses (measured in tens of thousands of US Dollars)
9. medianHouseValue: Median house value for households within a block (measured in US Dollars)
10. oceanProximity: Location of the house w.r.t ocean/sea

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
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import plot_tree

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,confusior
```

Reading data

In [2]:

```
df = pd.read_csv("/Users/HP/Desktop/housing1.csv")
df
```

Out[2]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_inco
0	-122.23	37.88	41	880	129.0	322	126	8.3
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3
2	-122.24	37.85	52	1467	190.0	496	177	7.2
3	-122.25	37.85	52	1274	235.0	558	219	5.6
4	-122.25	37.85	52	1627	280.0	565	259	3.8
...	...	...	...	...	...	...	...	

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_inco
20635	-121.09	39.48	25	1665	374.0	845	330	1.5
20636	-121.21	39.49	18	697	150.0	356	114	2.5
20637	-121.22	39.43	17	2254	485.0	1007	433	1.7
20638	-121.32	39.43	18	1860	409.0	741	349	1.8
20639	-121.24	39.37	16	2785	616.0	1387	530	2.3

20640 rows × 10 columns

In [3]:

```
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  int64  
 3   total_rooms         20640 non-null  int64  
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  int64  
 6   households          20640 non-null  int64  
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  int64  
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(4), int64(5), object(1)
memory usage: 1.6+ MB
```

In [4]:

```
df.describe().T
```

Out[4]:

	count	mean	std	min	25%	50%	75%	
longitude	20640.0	-119.569704	2.003532	-124.3500	-121.8000	-118.4900	-118.01000	
latitude	20640.0	35.631861	2.135952	32.5400	33.9300	34.2600	37.71000	
housing_median_age	20640.0	28.639486	12.585558	1.0000	18.0000	29.0000	37.00000	
total_rooms	20640.0	2635.763081	2181.615252	2.0000	1447.7500	2127.0000	3148.00000	3
total_bedrooms	20433.0	537.870553	421.385070	1.0000	296.0000	435.0000	647.00000	
population	20640.0	1425.476744	1132.462122	3.0000	787.0000	1166.0000	1725.00000	3
households	20640.0	499.539680	382.329753	1.0000	280.0000	409.0000	605.00000	
median_income	20640.0	3.870671	1.899822	0.4999	2.5634	3.5348	4.74325	
median_house_value	20640.0	206855.816909	115395.615874	14999.0000	119600.0000	179700.0000	264725.00000	50

checking for nulls + duplicated

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
longitude             0
latitude              0
```

```
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

- The feature having the 207 null values is a numerical feature
- Dropping out the nulls or replacing it with an appropriate strategy like their mean or median as it is a numerical feature
- Decided to drop out the nulls as we have a huge dataset contaning 20640 rows

In [6]:

```
df.dropna(inplace = True, axis = 0)
```

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20433 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20433 non-null  float64
 1   latitude            20433 non-null  float64
 2   housing_median_age  20433 non-null  int64
 3   total_rooms         20433 non-null  int64
 4   total_bedrooms      20433 non-null  float64
 5   population          20433 non-null  int64
 6   households          20433 non-null  int64
 7   median_income       20433 non-null  float64
 8   median_house_value  20433 non-null  int64
 9   ocean_proximity     20433 non-null  object
dtypes: float64(4), int64(5), object(1)
memory usage: 1.7+ MB
```

In [8]:

```
df.duplicated().sum()
```

Out[8]:

```
0
```

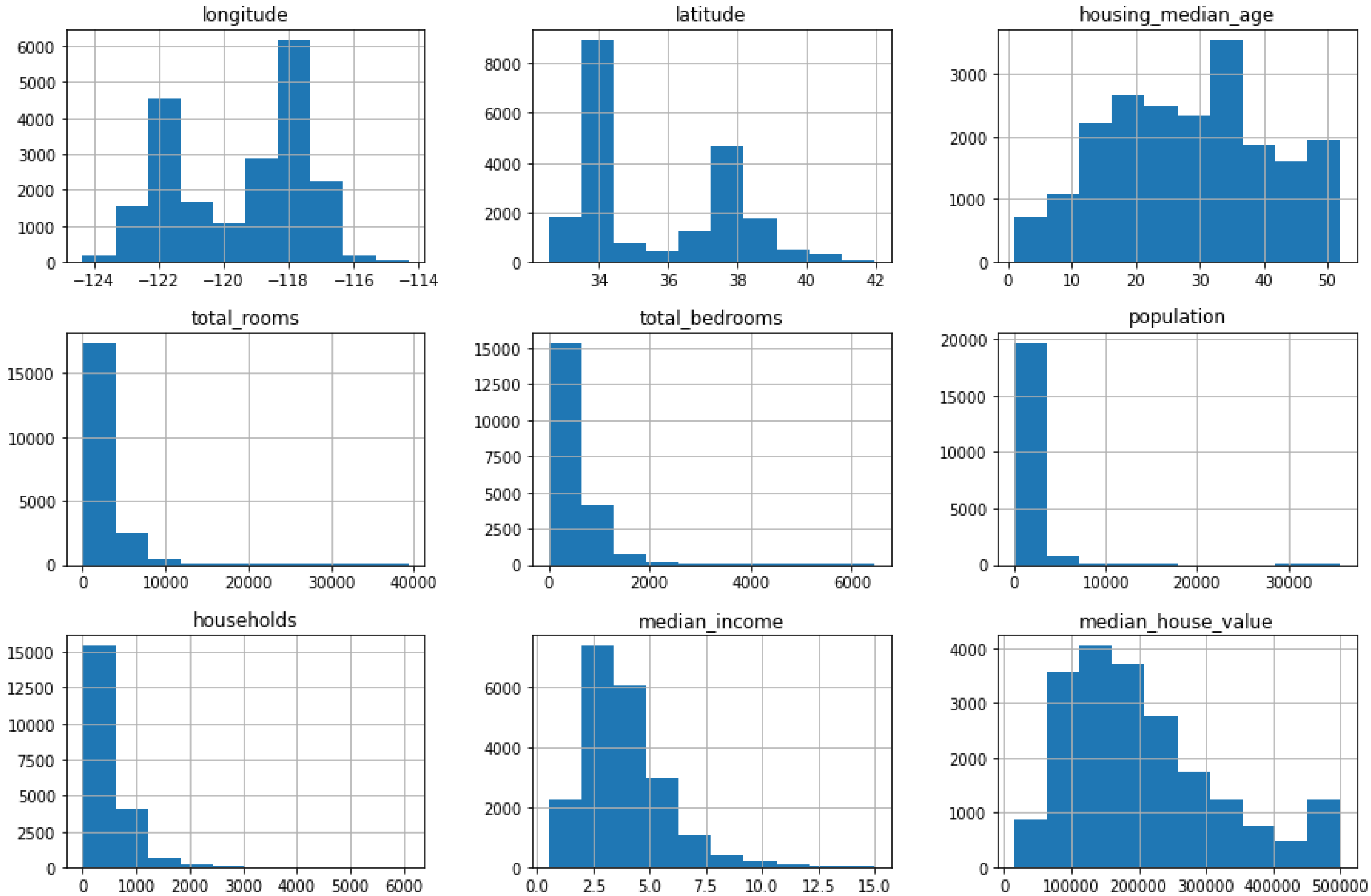
EDA

In [9]:

```
df.hist(figsize = (15,10))
```

Out[9]:

```
array([[<Axes: title={'center': 'longitude'}>,
        <Axes: title={'center': 'latitude'}>,
        <Axes: title={'center': 'housing_median_age'}>],
       [<Axes: title={'center': 'total_rooms'}>,
        <Axes: title={'center': 'total_bedrooms'}>,
        <Axes: title={'center': 'population'}>],
       [<Axes: title={'center': 'households'}>,
        <Axes: title={'center': 'median_income'}>,
        <Axes: title={'center': 'median_house_value'}>]], dtype=object)
```



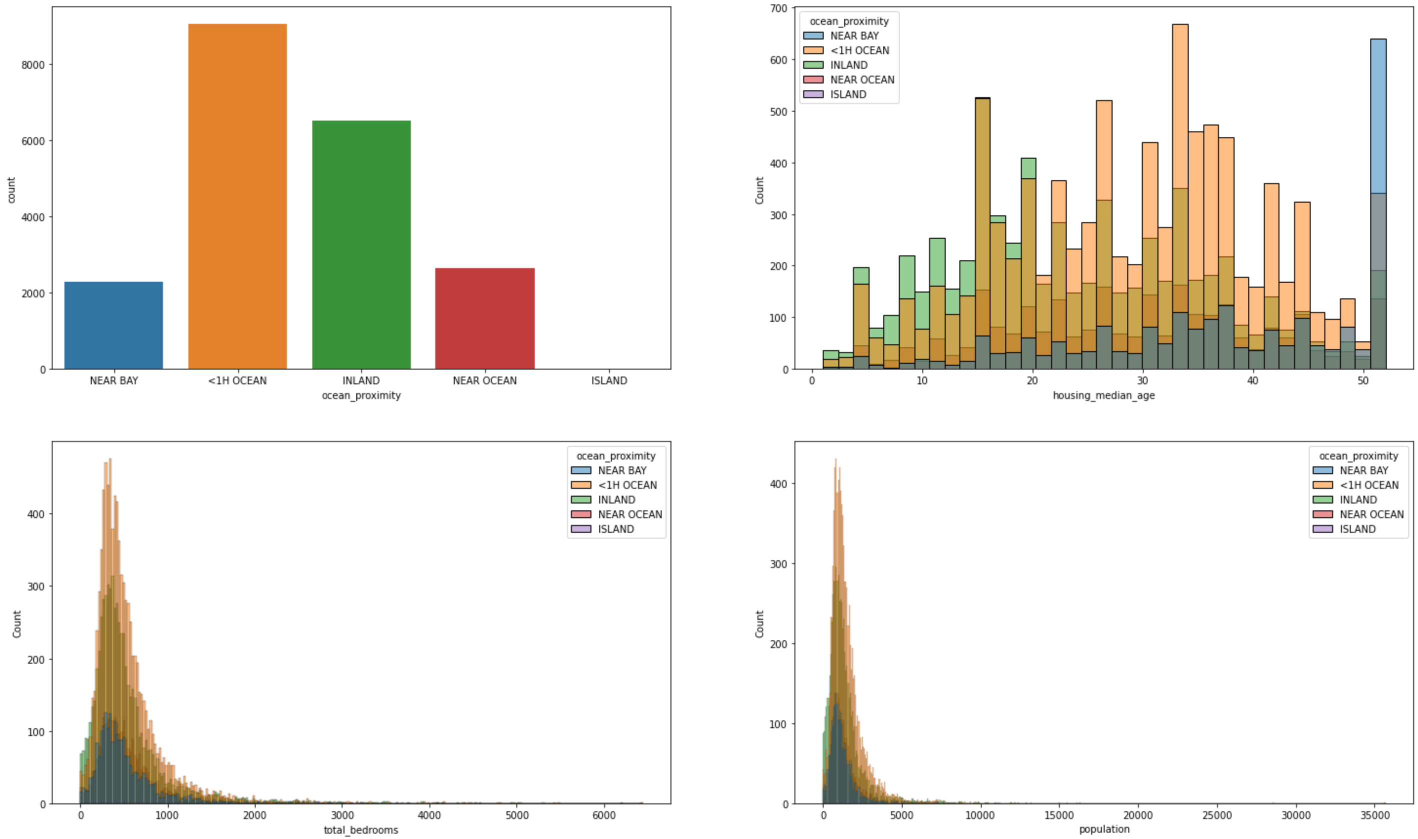
- These are the histograms of all the features which show out their distribution

In [10]:

```
plt.figure(figsize = (25,15))

# subplot 1
plt.subplot(2, 2, 1)
sns.countplot(x = df['ocean_proximity'] , data = df)

# subplot 2
plt.subplot(2, 2, 2)
sns.histplot(x = df['housing_median_age'], hue = df['ocean_proximity'])

# subplot 3
plt.subplot(2, 2, 3)
sns.histplot(x = df['total_bedrooms'], hue = df['ocean_proximity'])

# subplot 4
plt.subplot(2, 2, 4)
sns.histplot(x = df['population'], hue = df['ocean_proximity'])
```

Out[10]:

```
<Axes: xlabel='population', ylabel='Count'>
```



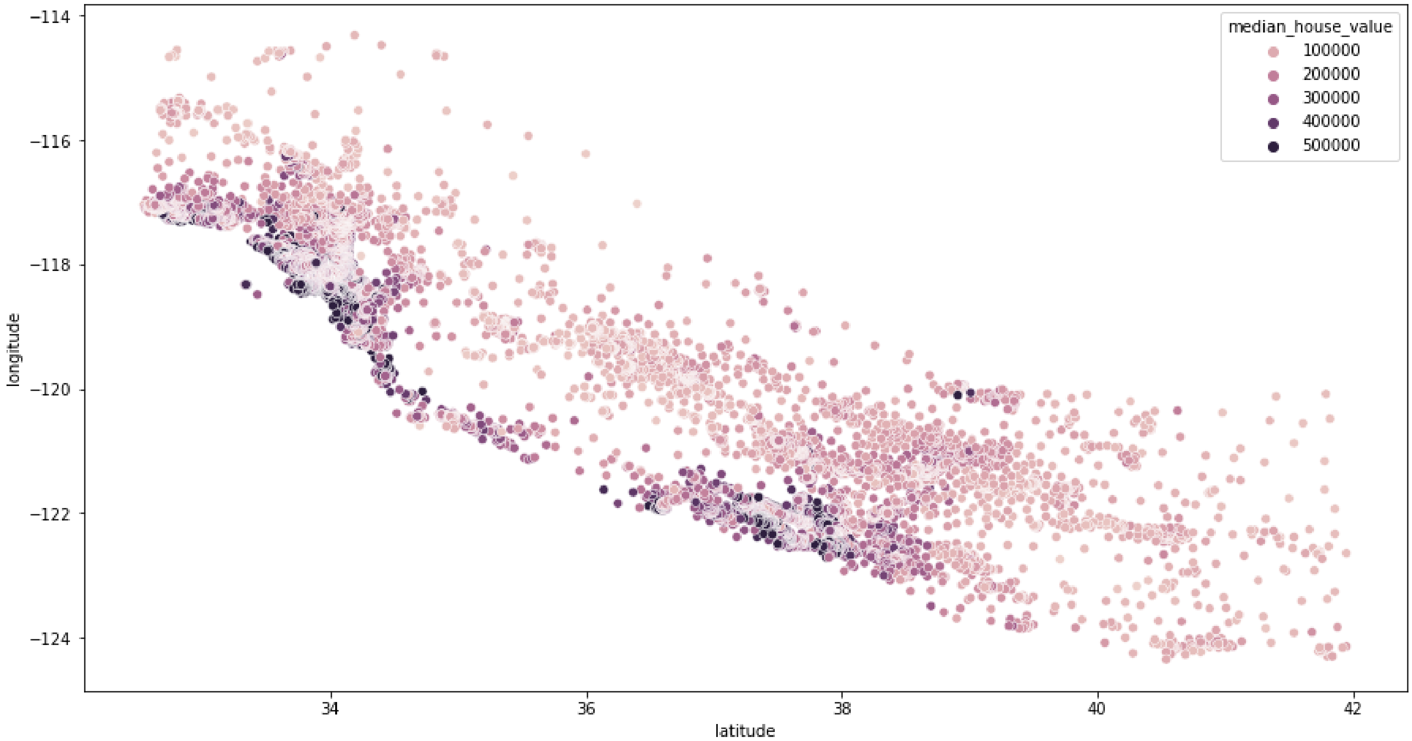
- There are more houses in our dataset that is less than 1 mile the ocean compared to rest of the labels
- Most Houses near the bay has the highest housing age with being more than 50 years
- The housing age of most of the houses less than 1 mile to ocean range from 20 to 45 years
- The housing age of most of the houses that is inland range from 5 to 20 years
- The housing age of most of the houses that is near the ocean range from 5 to 37 years
- For the total bedrooms, most of the range are from 200 to 800 bedrooms with the most counts going to houses less than 1 mile to ocean then to houses near the ocean
- For the population, most of the range are from 1000 to 2500 people with most counts goign to houses less than 1 mile to ocean then to houses near the ocean

In [11]:

```
plt.figure(figsize = (15,8))
sns.scatterplot(x = 'latitude', y = 'longitude', data = df, hue = 'median_house_value')
```

Out[11]:

```
<Axes: xlabel='latitude', ylabel='longitude'>
```



- Plotting a scatter plot with the longitude and latitude as it's axis with respect to the median_house _value can give us the map of CALIFORNIA
- As we can see , down on the left side is the sea and going up to the upper right side of the plot is the inland of california
- As we go near the sea , we find that the median house value increases compared to going deeper into the inland

Encoding of (ocean_proximity)

In [12]:

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

Out[12]:

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

- We find that this feature is a nominal one so it is best dealt with one hot encoding

In [13]:

```
df['ocean_proximity'].value_counts()
```

Out[13]:

```
<1H OCEAN     9034
INLAND        6496
NEAR OCEAN    2628
NEAR BAY      2270
ISLAND           5
Name: ocean_proximity, dtype: int64
```

In [14]:

```
pd.get_dummies(df['ocean_proximity'], drop_first = False)
```

Out[14]:

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	0	0	0	1	0
1	0	0	0	1	0
2	0	0	0	1	0

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
3	0	0	0	1	0
4	0	0	0	1	0
...	...	...	...	...	...
20635	0	1	0	0	0
20636	0	1	0	0	0
20637	0	1	0	0	0
20638	0	1	0	0	0
20639	0	1	0	0	0

20433 rows × 5 columns

In [15]:

```
df = df.join(pd.get_dummies(df['ocean_proximity'], drop_first = False)).drop(['ocean_proxi
df
```

Out[15]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_inco
0	-122.23	37.88	41	880	129.0	322	126	8.3
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3
2	-122.24	37.85	52	1467	190.0	496	177	7.2
3	-122.25	37.85	52	1274	235.0	558	219	5.6
4	-122.25	37.85	52	1627	280.0	565	259	3.8
...	...	...	...	...	...	...	...	
20635	-121.09	39.48	25	1665	374.0	845	330	1.5
20636	-121.21	39.49	18	697	150.0	356	114	2.5
20637	-121.22	39.43	17	2254	485.0	1007	433	1.7
20638	-121.32	39.43	18	1860	409.0	741	349	1.8
20639	-121.24	39.37	16	2785	616.0	1387	530	2.3

20433 rows × 14 columns

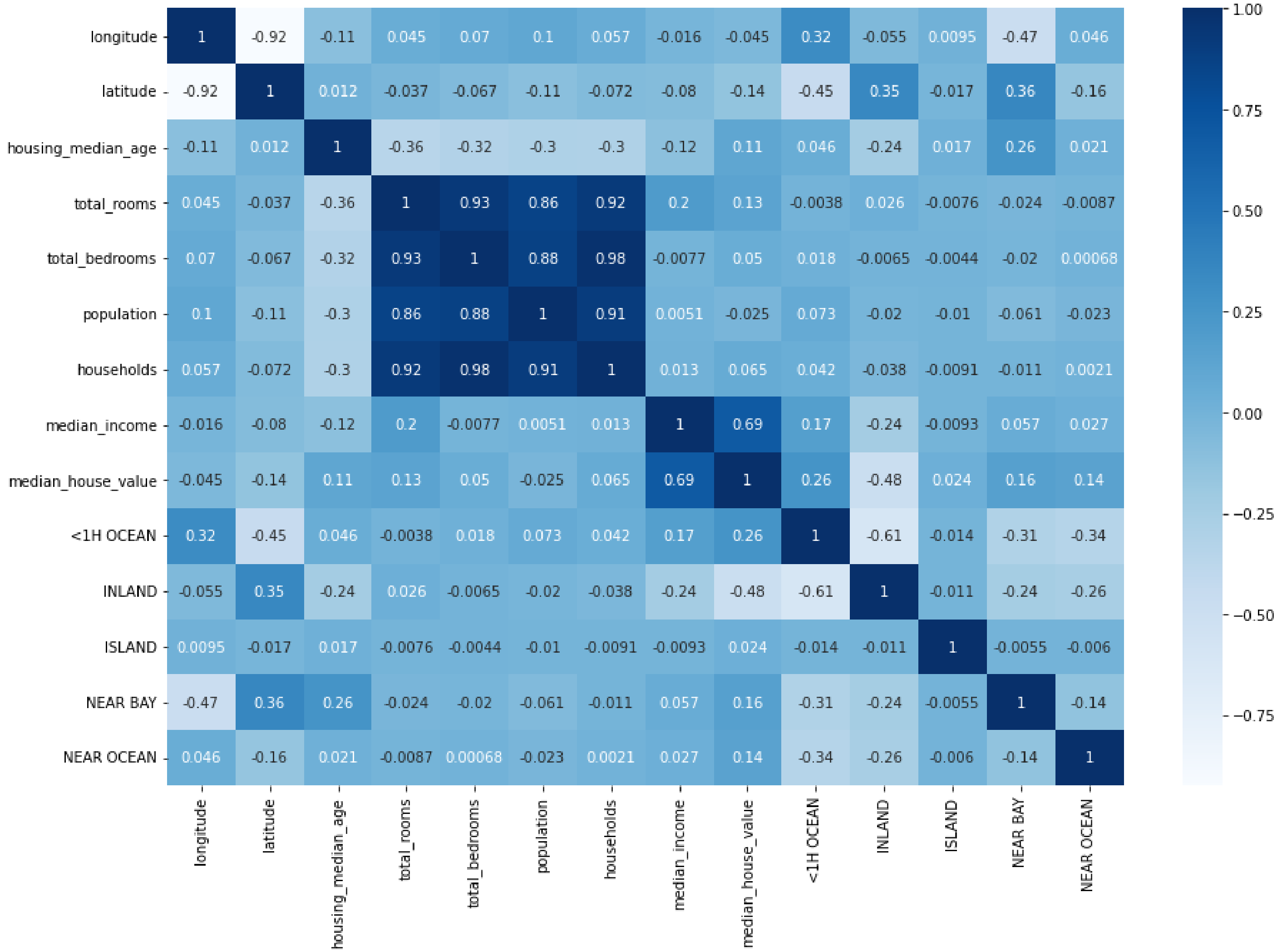
Correlation

In [16]:

```
corr = df.corr()
plt.figure(figsize = (15,10))
sns.heatmap(corr,annot = True,cmap = 'Blues')
```

Out[16]:

```
<Axes: >
```



- We can find that most of the features are not heavily correlated to each other except for the features of (total_bedrooms, total_rooms, population and households)
- This is great for our performance of the model
- No feature selection is needed as we just can ignore dropping them out as they are critical for the evaluation of the house price

Splitting of Data

In [17]:

```
x = df.drop(['median_house_value'] , axis = 1)
y = df['median_house_value']
```

In [18]:

```
x
```

Out[18]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_inco
0	-122.23	37.88	41	880	129.0	322	126	8.3
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3
2	-122.24	37.85	52	1467	190.0	496	177	7.2
3	-122.25	37.85	52	1274	235.0	558	219	5.6
4	-122.25	37.85	52	1627	280.0	565	259	3.8

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_inco
...	...	...	...	...	...	...	...	...
20635	-121.09	39.48	25	1665	374.0	845	330	1.5
20636	-121.21	39.49	18	697	150.0	356	114	2.5
20637	-121.22	39.43	17	2254	485.0	1007	433	1.7
20638	-121.32	39.43	18	1860	409.0	741	349	1.8
20639	-121.24	39.37	16	2785	616.0	1387	530	2.3

20433 rows × 13 columns

In [19]:

```
y
```

Out[19]:

```
0        452600
1        358500
2        352100
3        341300
4        342200
          ...
20635     78100
20636     77100
20637     92300
20638     84700
20639     89400
Name: median_house_value, Length: 20433, dtype: int64
```

train test split

In [20]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2, random_state =42

x_train,x_val,y_train,y_val = train_test_split(x_train,y_train,test_size = 0.2, random_sta
```

Scaling

In [21]:

```
scaler = StandardScaler()

x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)
x_test = scaler.transform(x_test)
```

- We just scaled the the input features
- We applied a fit_transform for the x_train
- For the rest, we just applied a transform function for them (x_val, x_test)

Linear Regression

In [22]:

```
## check overfitting

lr = LinearRegression()
```

```python
lr.fit(x_train,y_train)

x_train_pred = lr.predict(x_train)
train_score = r2_score(y_train,x_train_pred)
print(f'the train score is ={train_score}')

x_val_pred = lr.predict(x_val)
val_score = r2_score(y_val,x_val_pred)
print(f'the valid score is ={val_score}')

# scores are equal so no overfitting
```

```
the train score is =0.6426658495799197
the valid score is =0.6571835430345303
```

In [23]:

```python
y_pred = lr.predict(x_test)

MSE = mean_squared_error(y_test, y_pred)
MAE = mean_absolute_error(y_test, y_pred)
RMSE = np.sqrt(MSE)
r2_lr = r2_score(y_test, y_pred)

print(f'MSE = {MSE}')
print(f'MAE = {MAE}')
print(f'RMSE = {RMSE}')
print(f'r2 = {r2_lr}')
```

```
MSE = 4807231043.392106
MAE = 50472.786733200286
RMSE = 69334.19822419602
r2 = 0.6484703845161299
```

KNN

In [24]:

```python
# Use grid search to find best value
knn = KNeighborsRegressor()

params_grid = {'n_neighbors': [3,5,7,9,11,13,15,17,19]}

grid = GridSearchCV(
    knn,
    params_grid,
    cv = 5
)
grid.fit(x_train,y_train)

print(f'the best value of k = {grid.best_params_}')
```

```
the best value of k = {'n_neighbors': 9}
```

In [25]:

```python
## check overfitting

knn = KNeighborsRegressor(n_neighbors = 9)
knn.fit(x_train,y_train)

x_train_pred = knn.predict(x_train)
train_score = r2_score(y_train,x_train_pred)
print(f'the train score is ={train_score}')

x_val_pred = knn.predict(x_val)
val_score = r2_score(y_val,x_val_pred)
print(f'the valid score is ={val_score}')
```

```
# scores are equal so no overfitting
```

```
the train score is =0.7770115361386621
the valid score is =0.7358094971464155
```

In [26]:

```
y_pred = knn.predict(x_test)

MSE = mean_squared_error(y_test, y_pred)
MAE = mean_absolute_error(y_test, y_pred)
RMSE = np.sqrt(MSE)
r2_knn = r2_score(y_test, y_pred)

print(f'MSE = {MSE}')
print(f'MAE = {MAE}')
print(f'RMSE = {RMSE}')
print(f'r2 = {r2_knn}')
```

```
MSE = 3792620365.937477
MAE = 41357.19082184705
RMSE = 61584.25420460555
r2 = 0.7226639687420676
```

Random Forest

In [27]:

```
# Use grid search to find best value
rf = RandomForestRegressor(random_state = 42)

params_grid = {
    'max_depth':[3,4,5,6,7,8,9,10],
    'n_estimators':[20,50,70,100]
}

grid = GridSearchCV(
    rf,
    params_grid,
    cv = 5
)
grid.fit(x_train,y_train)

print(f'the best value of max_depth, n_estimators = {grid.best_params_}')
```

```
the best value of max_depth, n_estimators = {'max_depth': 10, 'n_estimators': 100}
```

In [28]:

```
## check overfitting

rf = RandomForestRegressor(max_depth = 10,n_estimators = 100)
rf.fit(x_train,y_train)

x_train_pred = rf.predict(x_train)
train_score = r2_score(y_train,x_train_pred)
print(f'the train score is ={train_score}')

x_val_pred = rf.predict(x_val)
val_score = r2_score(y_val,x_val_pred)
print(f'the valid score is ={val_score}')

# scores are approximately equal so no overfitting
```

```
the train score is =0.8631945828467154
the valid score is =0.7936770398793223
```

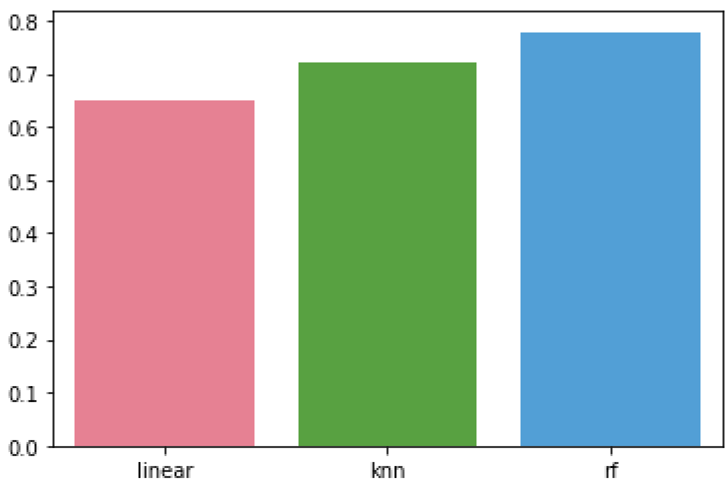
In [29]:
```
y_pred = rf.predict(x_test)

MSE = mean_squared_error(y_test, y_pred)
MAE = mean_absolute_error(y_test, y_pred)
RMSE = np.sqrt(MSE)
r2_rf = r2_score(y_test, y_pred)

print(f'MSE = {MSE}')
print(f'MAE = {MAE}')
print(f'RMSE = {RMSE}')
print(f'r2 = {r2_rf}')
```

```
MSE = 3031635170.714077
MAE = 37220.53421525975
RMSE = 55060.28669298841
r2 = 0.778311144975361
```

Comparsion of Models Evaluation (r2_score)

In [30]:
```
#comparison of all models r2 score
models_names = ['linear','knn','rf']
models_scores = [r2_lr,r2_knn,r2_rf]

sns.barplot(x = models_names, y = models_scores, data =df, palette = 'husl')
```

Out[30]: <Axes: >



- random forest scored the highest r2_score
- Knn model came in second
- lastly was the linear regression

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```