Deep Learning in Real Estate Prediction: An Empirical Study on California House Prices

Abstract:

Machine learning has become increasingly prevalent in the real estate industry for predicting house values and providing key indicators for investors in making informed decisions on the market trends. In this research, by integrating computer science and artificial intelligence (AI), various machine learning (ML) algorithms and data features were tested to determine which procedures yielded the highest accuracy. Based on this research, it will assist real estate investors with predictive analytics for the pricing trend and better market segmentations, risk assessments, demand forecasting and portfolio optimization. The same process can also be applied and tested for the Short Term Rental (STR) market for Airbnb investors in the current dynamic real estate market. The imported dataset encompasses variables from the dynamic California housing market, layered with additional geographic parameters. Linear regression, support vector machines with linear, polynomial, and radial basis function kernels, and deep neural networks were reviewed and tested. To assess the accuracy of the regression models, RMSE was used as an evaluation metric. Upon analyzing the RMSE results, the SVR with the RBF kernel exhibited the lowest error values compared to the performance of other models. This indicates the suitability of this model for the regression task and highlights the impact of model hyperparameter choice on task performance. The exceptional performance of the RBF kernel makes it a valuable candidate for real-world applications and future house price predictions in the volatile real estate market. These findings lay the foundation for model optimization, feature engineering, and further investigation into the characteristics of the dataset and alternative model architectures.

Part I: Dataset

1.1 Understand dataset

```
# step 1: Import Dataset
import pandas as pd
import numpy as np

file_name = "housing.csv"
housing = pd.read_csv(file_name)
housing.head()
```

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

Next steps: View recommended plots

```
housing.shape # there are 20,640 observations on 9 variables.
```

```
(20640, 10)
```

```
# Obtain features/variables
housing.columns
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity'],
      dtype='object')
```

```
# Obtain the types of features
housing.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
```

```
# Have a look at Categorical data
housing["ocean_proximity"].unique()
```

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

```
# Count value of categorical data
housing["ocean_proximity"].value_counts()
```

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

1.2 Feature statsitics

```
# Feature statsitics shows a summary of the numerical attributes
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_v
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.0C
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.81
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.61
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.0C
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.0C
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.0C
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.0C
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.0C

```
# Describe statistics of dependent variable
housing.describe()["median_house_value"]
```

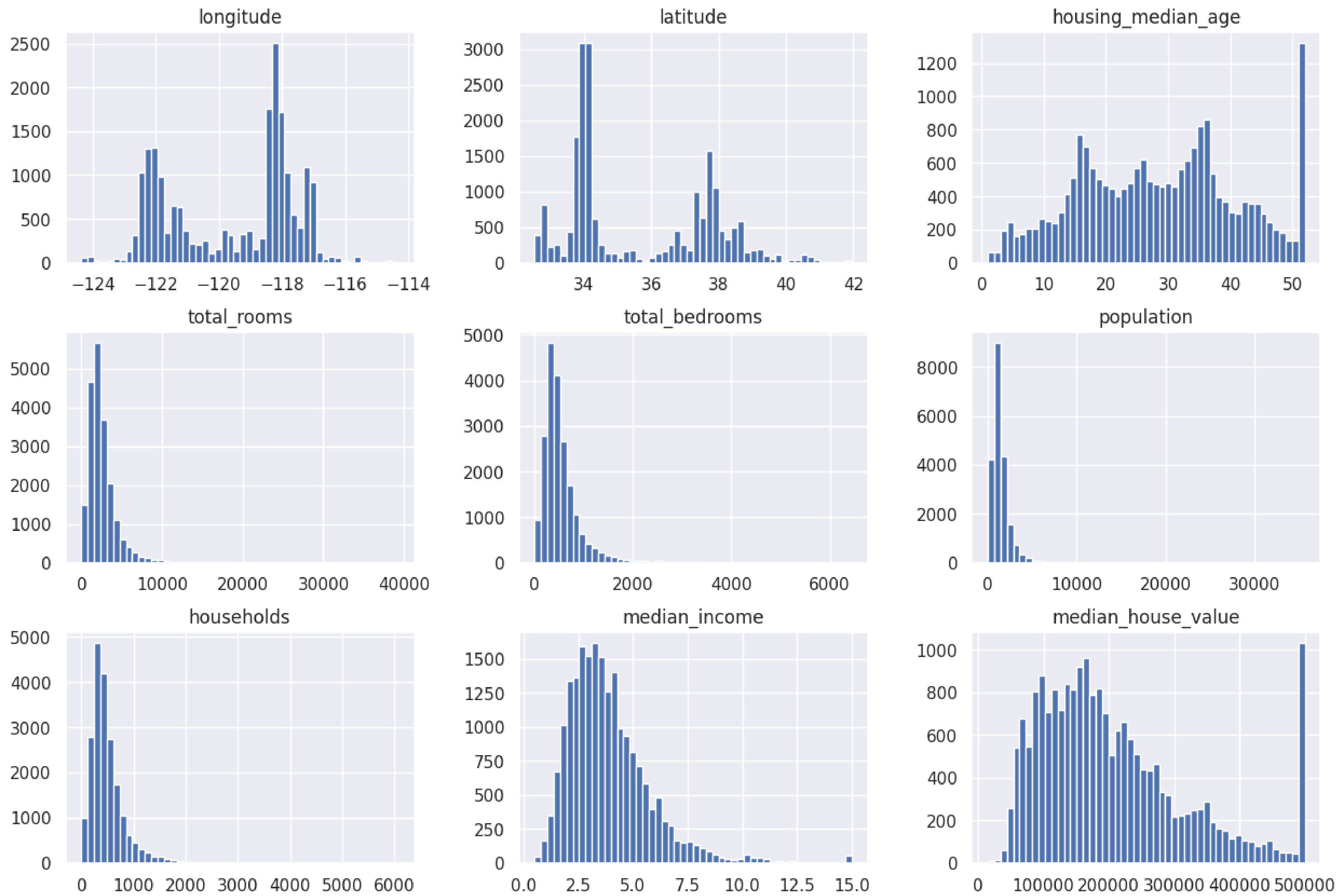
```
count     20640.000000
mean     206855.816909
std      115395.615874
min       14999.000000
25%      119600.000000
50%      179700.000000
75%      264725.000000
max      500001.000000
Name: median_house_value, dtype: float64
```

```
# Identify missing value
housing.isnull().sum()
```

```
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
```

1.3 Histgram / Distribution and Outliers

```
# plot a histogram for each numerical attribute
%matplotlib inline
import matplotlib.pyplot as plt

housing.hist(bins=50, figsize=(15,10))
plt.show()
```



1.4 Correlation

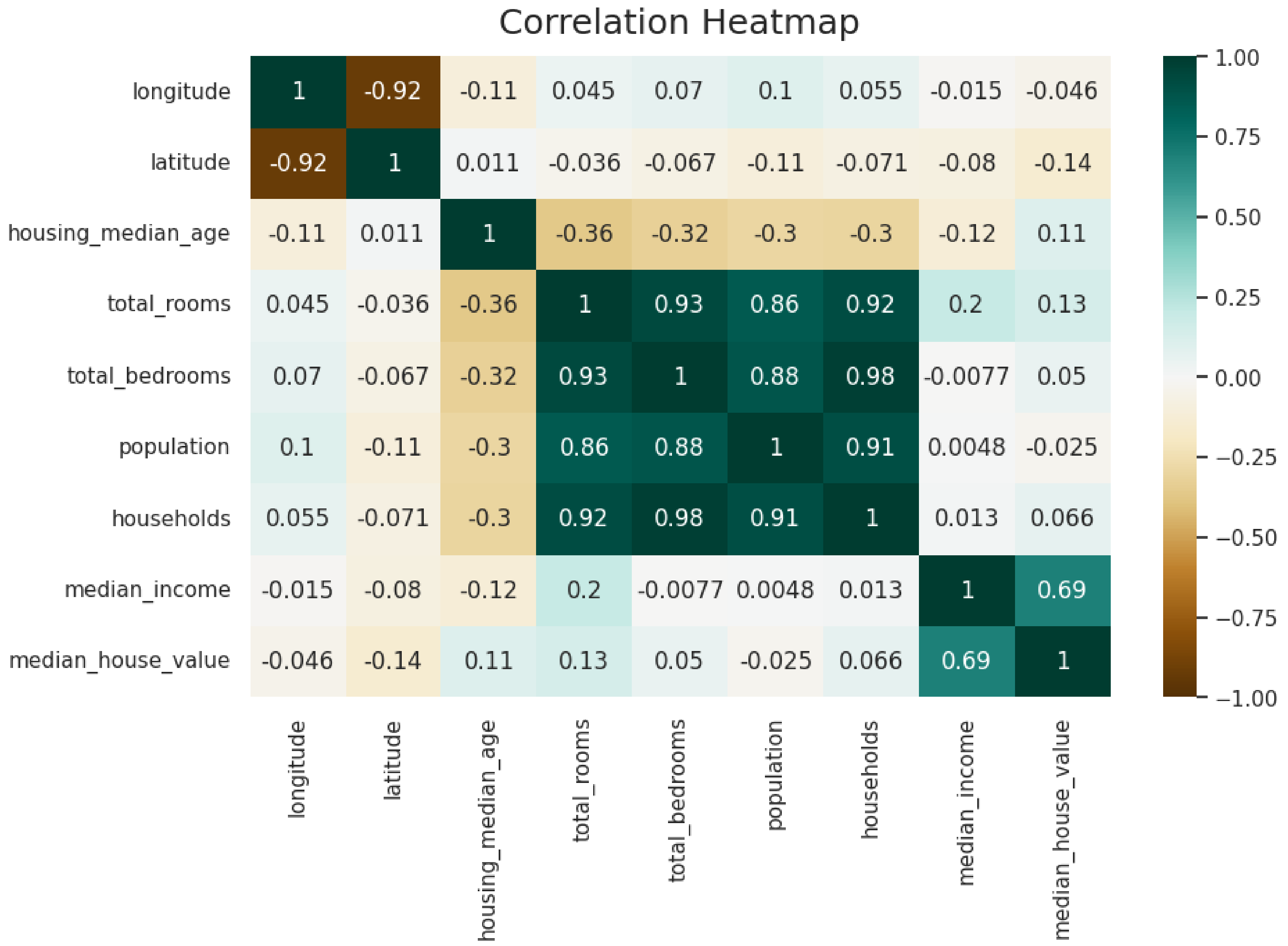
```
# Looking for Correlations
corr_matrix = housing.drop(columns=["ocean_proximity"]).corr()
corr_matrix
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_
longitude	1.000000	-0.924664	-0.108197	0.044568	0.069608	0.099773	0.055310	-0.015176	
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.066983	-0.108785	-0.071035	-0.079809	
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.320451	-0.296244	-0.302916	-0.119034	
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.930380	0.857126	0.918484	0.198050	
total_bedrooms	0.069608	-0.066983	-0.320451	0.930380	1.000000	0.877747	0.979728	-0.007723	
population	0.099773	-0.108785	-0.296244	0.857126	0.877747	1.000000	0.907222	0.004834	
households	0.055310	-0.071035	-0.302916	0.918484	0.979728	0.907222	1.000000	0.013033	
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.007723	0.004834	0.013033	1.000000	
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.049686	-0.024650	0.065843	0.688075	

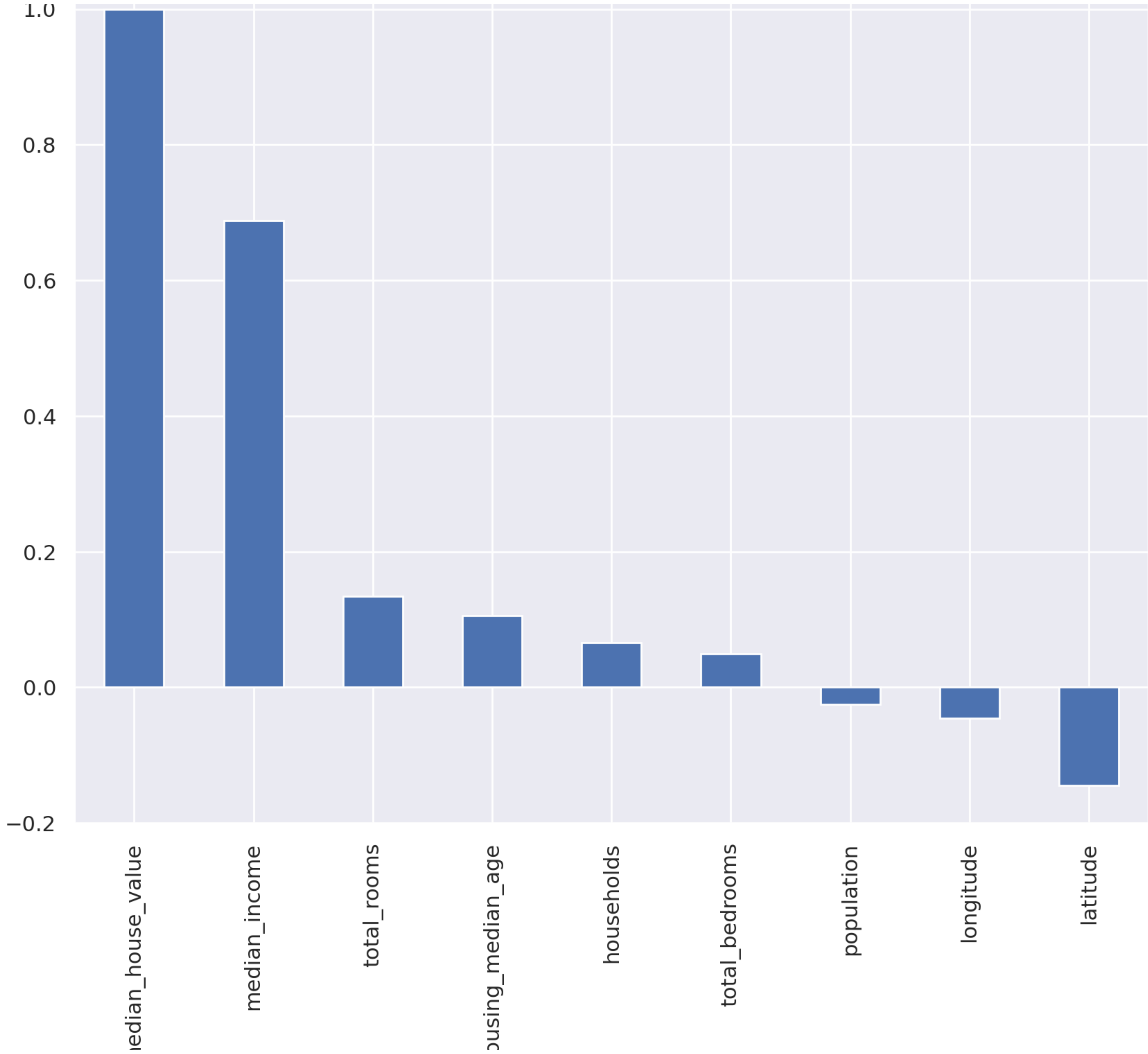
Next steps: View recommended plots

```
import seaborn as sns

plt.figure(figsize=(10, 6))
heatmap = sns.heatmap(housing.drop(columns=["ocean_proximity"]).corr(), vmin=-1, vmax=1, annot=True, cmap='BrBG')
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':18}, pad=12)
```

```
Text(0.5, 1.0, 'Correlation Heatmap')
```




```
sns.set()
plt.figure(figsize=(10,8), dpi=200)

housing.drop(columns=["ocean_proximity"]).corr()['median_house_value'].sort_values(ascending = False).plot(kind='bar')
```



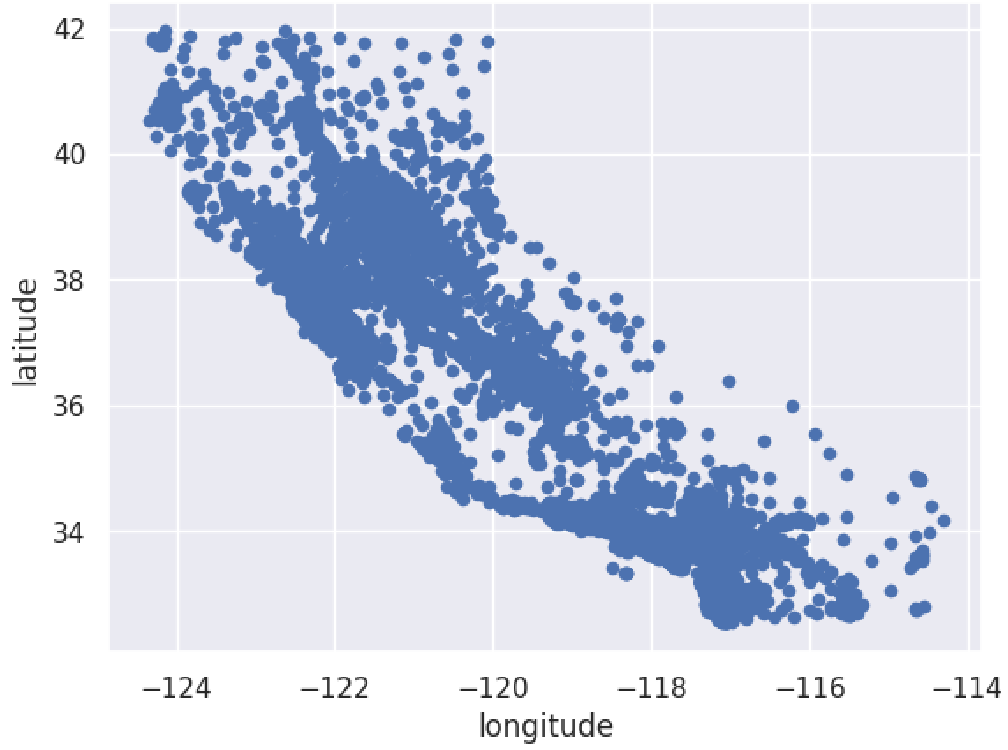

```
corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

1.5 Geographical Information

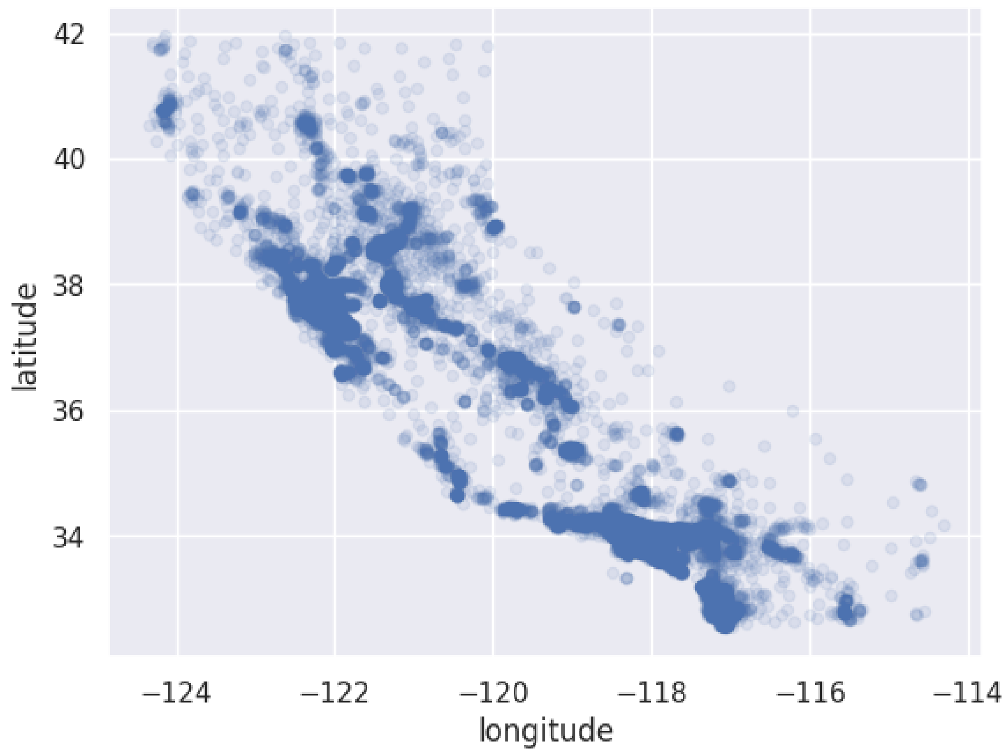
```
# Visualizing Geographical Data
housing.plot(kind="scatter", x="longitude", y="latitude")
```

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



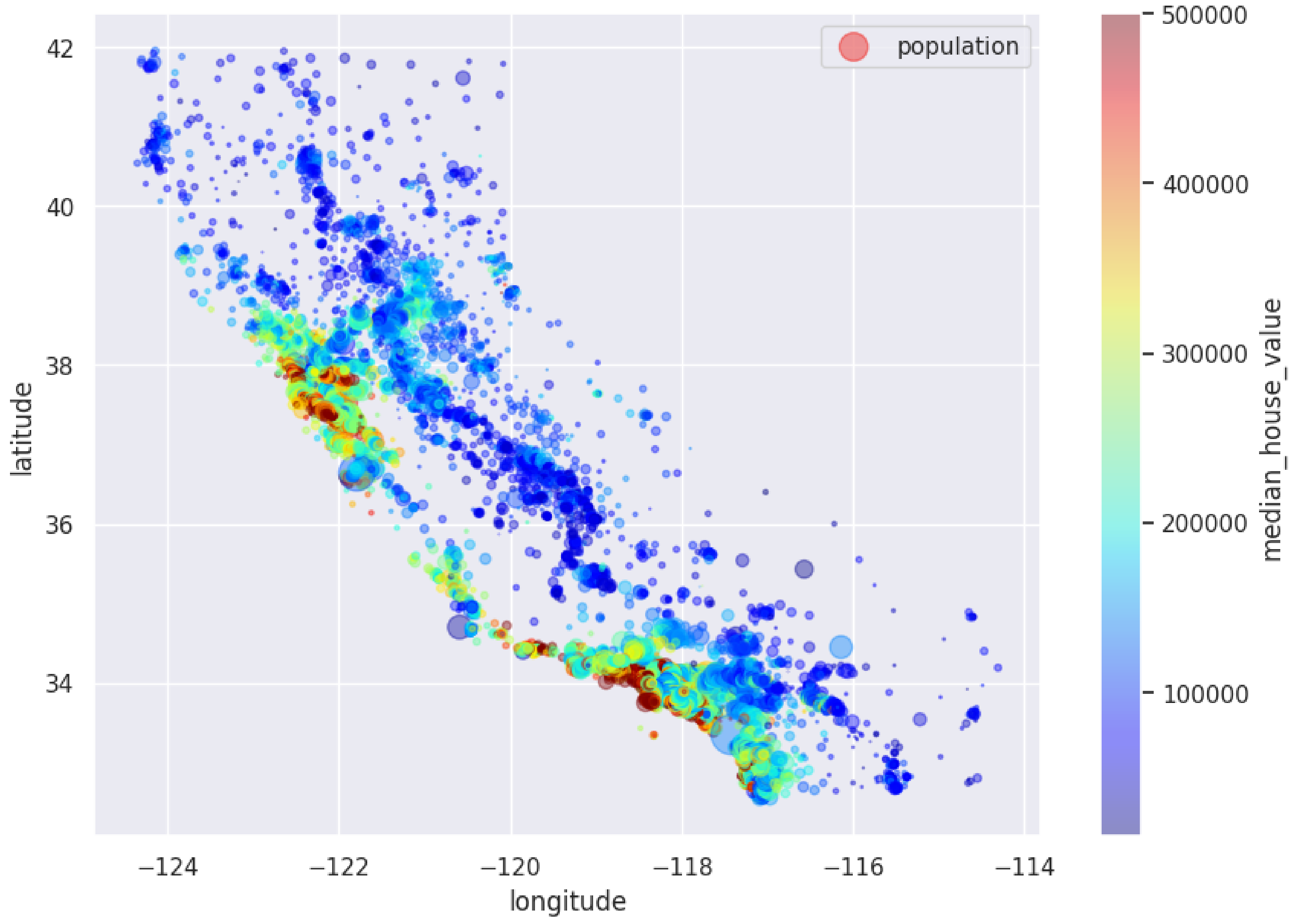

```
# Observe the density of observation
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

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




```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
             s=housing["population"]/100, label="population", figsize=(10,7),
             c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,)
plt.legend()
```

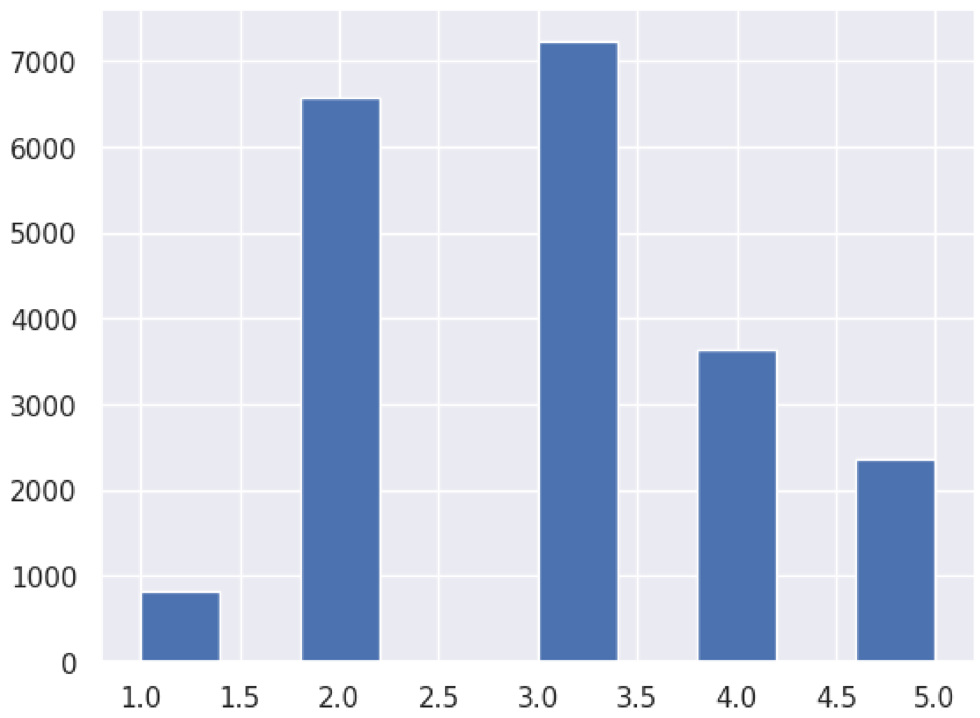
```
<matplotlib.legend.Legend at 0x79fda8bd6a70>
```



1.6 Feature Engineering - Adding new features

```
# Add a new categorical feature in order to split the dataset properly
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                            labels=[1, 2, 3, 4, 5])
housing["income_cat"].hist()
```

```
<Axes: >
```




```
# Count value of different categories
housing.income_cat.value_counts()/ len(housing.income_cat)
```

```
3    0.350581
2    0.318847
4    0.176308
5    0.114438
1    0.039826
Name: income_cat, dtype: float64
```

```
# Experimenting with Attribute Combinations
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

```
corr_matrix = housing.drop(columns=["ocean_proximity"]).corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
<ipython-input-106-ff52c9985b1f>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  corr_matrix = housing.drop(columns=["ocean_proximity"]).corr()
```

```
median_house_value          1.000000
median_income               0.688075
rooms_per_household         0.151948
total_rooms                 0.134153
housing_median_age          0.105623
households                  0.065843
total_bedrooms              0.049686
population_per_household   -0.023737
population                 -0.024650
longitude                  -0.045967
latitude                   -0.144160
bedrooms_per_room          -0.255880
Name: median_house_value, dtype: float64
```

Part II: Data Preprocessing - Prepare Data for Machine Learning Algorithm

```
housing.columns
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity', 'income_cat',
       'rooms_per_household', 'bedrooms_per_room', 'population_per_household'],
      dtype='object')
```

```
category_cols1 = ["ocean_proximity","income_cat"]
category_cols2 = ["income_cat"]
numerical_cols = ['longitude', 'latitude', 'housing_median_age', 'total_rooms',
                  'total_bedrooms', 'population', 'households', 'median_income',
                  'rooms_per_household', 'bedrooms_per_room', 'population_per_household']
target = "median_house_value"

# Validate number of attributes
assert len(category_cols1) + len(numerical_cols) + 1 == housing.shape[1]
```

2.1 Handling Text and Categorical Attributes

```
from sklearn.preprocessing import OneHotEncoder

housing_cat = housing[["ocean_proximity"]]
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

```
<20640x5 sparse matrix of type '<class 'numpy.float64'>'
    with 20640 stored elements in Compressed Sparse Row format>
```

```
# Convert the sparse matrix to  Numpy array
housing_cat_1hot.toarray()
```

```
array([[0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       ...,
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

```
# Get the list of categories using the encoder’s categories_ instance variable:
cat_encoder.categories_
```

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

```
category_df = pd.DataFrame(housing_cat_1hot.toarray(), columns = ["Ocean_Proximity_"+i for i in cat_encoder.categories_])
category_df.head()
```

	Ocean_Proximity_<1H OCEAN	Ocean_Proximity_INLAND	Ocean_Proximity_ISLAND	Ocean_Proximity_NEAR BAY	Ocean_Proximity_NEAR OCEAN
0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	1.0	0.0

Next steps: View recommended plots

2.2 Handling Numerical Data

```
housing_num = housing[numerical_cols]
housing_num.sample(5)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	rooms_per_household
17813	-121.85	37.39	15.0	8748.0	1547.0	4784.0	1524.0	5.8322	5.740157
20509	-121.53	38.60	25.0	5154.0	1105.0	3196.0	1073.0	2.7566	4.803355
20011	-119.12	36.05	27.0	1575.0	321.0	1063.0	317.0	2.1477	4.968454
11833	-120.18	39.28	14.0	10098.0	1545.0	701.0	254.0	4.0819	39.755906
1678	-122.32	38.06	4.0	7999.0	1611.0	3596.0	1396.0	5.0969	5.729943

```
housing_num = housing_num.fillna(housing_num.median())
housing_num.isnull().sum()
```

```
longitude                   0
latitude                    0
housing_median_age          0
total_rooms                 0
total_bedrooms              0
population                  0
households                  0
median_income               0
rooms_per_household         0
bedrooms_per_room           0
population_per_household    0
dtype: int64
```

2.3 Feature Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
housing_num_scale = scaler.fit_transform(housing_num)
```

```
numerical_df = pd.DataFrame(housing_num_scale, columns = housing_num.columns)
numerical_df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	rooms_per_household	bec
0	-1.327835	1.052548	0.982143	-0.804819	-0.972476	-0.974429	-0.977033	2.344766	0.628559	
1	-1.322844	1.043185	-0.607019	2.045890	1.357143	0.861439	1.669961	2.332238	0.327041	
2	-1.332827	1.038503	1.856182	-0.535746	-0.827024	-0.820777	-0.843637	1.782699	1.155620	
3	-1.337818	1.038503	1.856182	-0.624215	-0.719723	-0.766028	-0.733781	0.932968	0.156966	
4	-1.337818	1.038503	1.856182	-0.462404	-0.612423	-0.759847	-0.629157	-0.012881	0.344711	

Next steps: View recommended plots

```
numerical_df.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	rooms_per_h
count	2.064000e+04	2.064000e+04	2.064000e+04	2.064000e+04	2.064000e+04	2.064000e+04	2.064000e+04	2.064000e+04	2.06
mean	-8.526513e-15	-1.079584e-15	5.508083e-18	3.201573e-17	-9.363741e-17	-1.101617e-17	6.885104e-17	6.609700e-17	6.6
std	1.000024e+00	1.000024e+00	1.000024e+00	1.000024e+00	1.000024e+00	1.000024e+00	1.000024e+00	1.000024e+00	1.00
min	-2.385992e+00	-1.447568e+00	-2.196180e+00	-1.207283e+00	-1.277688e+00	-1.256123e+00	-1.303984e+00	-1.774299e+00	-1.85
25%	-1.113209e+00	-7.967887e-01	-8.453931e-01	-5.445698e-01	-5.718868e-01	-5.638089e-01	-5.742294e-01	-6.881186e-01	-3.9
50%	5.389137e-01	-6.422871e-01	2.864572e-02	-2.332104e-01	-2.428309e-01	-2.291318e-01	-2.368162e-01	-1.767951e-01	-8.0
75%	7.784964e-01	9.729566e-01	6.643103e-01	2.348028e-01	2.537334e-01	2.644949e-01	2.758427e-01	4.593063e-01	2.5
max	2.625280e+00	2.958068e+00	1.856182e+00	1.681558e+01	1.408779e+01	3.025033e+01	1.460152e+01	5.858286e+00	5.51

```
# Combine numerical data with categorical data
housing_new = pd.concat([numerical_df, category_df, housing["income_cat"], housing[target]], axis=1)
housing_new.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	rooms_per_household	bed
0	-1.327835	1.052548	0.982143	-0.804819	-0.972476	-0.974429	-0.977033	2.344766	0.628559	
1	-1.322844	1.043185	-0.607019	2.045890	1.357143	0.861439	1.669961	2.332238	0.327041	
2	-1.332827	1.038503	1.856182	-0.535746	-0.827024	-0.820777	-0.843637	1.782699	1.155620	
3	-1.337818	1.038503	1.856182	-0.624215	-0.719723	-0.766028	-0.733781	0.932968	0.156966	
4	-1.337818	1.038503	1.856182	-0.462404	-0.612423	-0.759847	-0.629157	-0.012881	0.344711	

Next steps: View recommended plots

```
housing_new.columns
```

```
Index([                     'longitude',                      'latitude',
                   'housing_median_age',                   'total_rooms',
                       'total_bedrooms',                    'population',
                           'households',                 'median_income',
                  'rooms_per_household',              'bedrooms_per_room',
             'population_per_household',  ('Ocean_Proximity_<1H OCEAN',),
            ('Ocean_Proximity_INLAND',),      ('Ocean_Proximity_ISLAND',),
          ('Ocean_Proximity_NEAR BAY',), ('Ocean_Proximity_NEAR OCEAN',),
                           'income_cat',              'median_house_value'],
      dtype='object')
```

```
housing_new.isnull().sum()
```

```
longitude                        0
latitude                         0
housing_median_age               0
total_rooms                      0
total_bedrooms                   0
population                       0
households                       0
median_income                    0
rooms_per_household              0
bedrooms_per_room                0
population_per_household         0
(Ocean_Proximity_<1H OCEAN,)     0
(Ocean_Proximity_INLAND,)        0
(Ocean_Proximity_ISLAND,)        0
(Ocean_Proximity_NEAR BAY,)      0
(Ocean_Proximity_NEAR OCEAN,)    0
income_cat                       0
median_house_value               0
dtype: int64
```

Part III: Data split

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

data4train,data4test = train_test_split(housing_new, test_size=0.2, random_state=42)
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state = 42)


for train_index, test_index in split.split(housing_new, housing['income_cat']):
    train_set = housing_new.loc[train_index]
    test_set  = housing_new.loc[test_index]
```

```
housing.income_cat.value_counts()/ len(housing.income_cat)
```

```
3    0.350581
2    0.318847
4    0.176308
5    0.114438
1    0.039826
Name: income_cat, dtype: float64
```

```
def table4income_cat(dataset,df,label):
    df[label]=pd.Series(dataset['income_cat'].value_counts()/len(dataset['income_cat']))
    return df

df = pd.DataFrame()
df = table4income_cat(train_set,df,'All_set')
df = table4income_cat(train_set,df,'train_set_Shuff')
df = table4income_cat(test_set,df,'test_set_Shuff')
df = table4income_cat(data4train,df,'train_set_split')
df = table4income_cat(data4test,df,'test_set_split')
df
```

	All_set	train_set_Shuff	test_set_Shuff	train_set_split	test_set_split
3	0.350594	0.350594	0.350533	0.348595	0.358527
2	0.318859	0.318859	0.318798	0.317466	0.324370
4	0.176296	0.176296	0.176357	0.178537	0.167393
5	0.114462	0.114462	0.114341	0.115673	0.109496
1	0.039789	0.039789	0.039971	0.039729	0.040213

Next steps: View recommended plots

```
# We use train_set and test_set
X_train = train_set.drop("median_house_value",axis=1)
y_train = train_set["median_house_value"].copy()
X_test = test_set.drop("median_house_value",axis=1)
y_test = test_set["median_house_value"].copy()
```

```
X_train.shape,y_train.shape, X_test.shape,y_test.shape
```

```
((16512, 17), (16512,), (4128, 17), (4128,))
```

Part IV: Train Model

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

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.optimizers import Adam
import keras
```

Model 1 -- Linear Regression

```
## model 1 -- Linear Regression
linear_model = LinearRegression()
linear_model.fit(X_train.to_numpy(), y_train.to_numpy())

linear_model_y_train_predict = linear_model.predict(X_train.to_numpy()).reshape(-1,1)
linear_model_y_test_predict = linear_model.predict(X_test.to_numpy()).reshape(-1,1)
```

Model 2 -- Support Vector Machine

```
# model 2a -- Support vector regression with linear kernal
svm2a  = svm.SVR(kernel = "linear", C =10000)
svm2a.fit(X_train.to_numpy(), y_train.to_numpy())

svm2a_y_train_predict = svm2a.predict(X_train.to_numpy()).reshape(-1,1)
svm2a_y_test_predict = svm2a.predict(X_test.to_numpy()).reshape(-1,1)
```

```
# model 2b -- Support vector regression with polynomial kernal
svm2b  = svm.SVR(kernel = "poly", C =10000)
svm2b.fit(X_train.to_numpy(), y_train.to_numpy())

svm2b_y_train_predict = svm2b.predict(X_train.to_numpy()).reshape(-1,1)
svm2b_y_test_predict = svm2b.predict(X_test.to_numpy()).reshape(-1,1)
```

```
# model 2c -- Support vector regression with rbf
svm2c  = svm.SVR(kernel = "rbf", C =50000)
svm2c.fit(X_train.to_numpy(), y_train.to_numpy())

svm2c_y_train_predict = svm2c.predict(X_train.to_numpy()).reshape(-1,1)
svm2c_y_test_predict = svm2c.predict(X_test.to_numpy()).reshape(-1,1)
```

Model 3 -- Deep Neural Network

```
# model 3 -- deep neural network
dnn = Sequential()
dnn.add(Dense(128,activation='relu'))
dnn.add(Dense(64,activation='relu'))
dnn.add(Dense(32,activation='relu'))
dnn.add(Dense(16,activation='relu'))
dnn.add(Dense(1))
dnn.compile(optimizer='Adam',loss='mse')

callback = keras.callbacks.EarlyStopping(monitor='val_loss',patience=3)

dnn.fit(x=X_train,y=y_train,
          validation_data=(X_test,y_test),
          batch_size=128,epochs=200,callbacks=[callback],
         )
dnn.summary()
```

```
129/129 [==============================] - 1s 4ms/step - loss: 3804526080.0000 - val_loss: 3764439552.0000
Epoch 49/200
129/129 [==============================] - 1s 4ms/step - loss: 3799829504.0000 - val_loss: 3724913408.0000
Epoch 50/200
129/129 [==============================] - 1s 5ms/step - loss: 3791100672.0000 - val_loss: 3704045568.0000
Epoch 51/200
129/129 [==============================] - 1s 5ms/step - loss: 3783483904.0000 - val_loss: 3718940160.0000
Epoch 52/200
129/129 [==============================] - 1s 4ms/step - loss: 3774644224.0000 - val_loss: 3712376576.0000
Epoch 53/200
129/129 [==============================] - 0s 3ms/step - loss: 3771256576.0000 - val_loss: 3695436032.0000
Epoch 54/200
129/129 [==============================] - 0s 3ms/step - loss: 3768870144.0000 - val_loss: 3693460736.0000
Epoch 55/200
129/129 [==============================] - 0s 3ms/step - loss: 3766198784.0000 - val_loss: 3685992192.0000
Epoch 56/200
129/129 [==============================] - 0s 3ms/step - loss: 3756240896.0000 - val_loss: 3681807872.0000
Epoch 57/200
129/129 [==============================] - 0s 3ms/step - loss: 3752671488.0000 - val_loss: 3682544384.0000
Epoch 58/200
129/129 [==============================] - 0s 3ms/step - loss: 3748231680.0000 - val_loss: 3694468864.0000
Epoch 59/200
129/129 [==============================] - 0s 3ms/step - loss: 3743673600.0000 - val_loss: 3675799808.0000
Epoch 60/200
129/129 [==============================] - 0s 3ms/step - loss: 3740482048.0000 - val_loss: 3663787520.0000
Epoch 61/200
129/129 [==============================] - 0s 3ms/step - loss: 3736025600.0000 - val_loss: 3651408640.0000
Epoch 62/200
129/129 [==============================] - 0s 3ms/step - loss: 3730261504.0000 - val_loss: 3650137856.0000
Epoch 63/200
129/129 [==============================] - 0s 3ms/step - loss: 3723343360.0000 - val_loss: 3668090368.0000
Epoch 64/200
129/129 [==============================] - 0s 3ms/step - loss: 3721384448.0000 - val_loss: 3635246848.0000
Epoch 65/200
129/129 [==============================] - 0s 3ms/step - loss: 3722064128.0000 - val_loss: 3653003008.0000
Epoch 66/200
129/129 [==============================] - 0s 3ms/step - loss: 3712539648.0000 - val_loss: 3639955968.0000
Epoch 67/200
129/129 [==============================] - 0s 3ms/step - loss: 3714742784.0000 - val_loss: 3638889984.0000
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 128)               2304

 dense_6 (Dense)             (None, 64)                8256

 dense 7 (Dense)             (None. 32)                2080
```