In [1]:

```
print ("Hello World")
```

```
Hello World
```

In [2]:

```
import pandas as pd
```

In [3]:

```
pd.read_csv("housing.csv")
```

Out[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	po
0	-122.23	37.88	41.0	880.0	129.0	
1	-122.22	37.86	21.0	7099.0	1106.0	
2	-122.24	37.85	52.0	1467.0	190.0	
3	-122.25	37.85	52.0	1274.0	235.0	
4	-122.25	37.85	52.0	1627.0	280.0	
...	...	...	...	...	...	
20635	-121.09	39.48	25.0	1665.0	374.0	
20636	-121.21	39.49	18.0	697.0	150.0	
20637	-121.22	39.43	17.0	2254.0	485.0	
20638	-121.32	39.43	18.0	1860.0	409.0	
20639	-121.24	39.37	16.0	2785.0	616.0	

20640 rows × 10 columns

In [4]:

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

In [5]:

```
housing.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-122.23	37.88	41.0	880.0	129.0	32
1	-122.22	37.86	21.0	7099.0	1106.0	240
2	-122.24	37.85	52.0	1467.0	190.0	49
3	-122.25	37.85	52.0	1274.0	235.0	55
4	-122.25	37.85	52.0	1627.0	280.0	56

In [6]:

```
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
```

In [7]:

```
housing["ocean_proximity"].value_counts()
```

Out[7]:

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

In [8]:

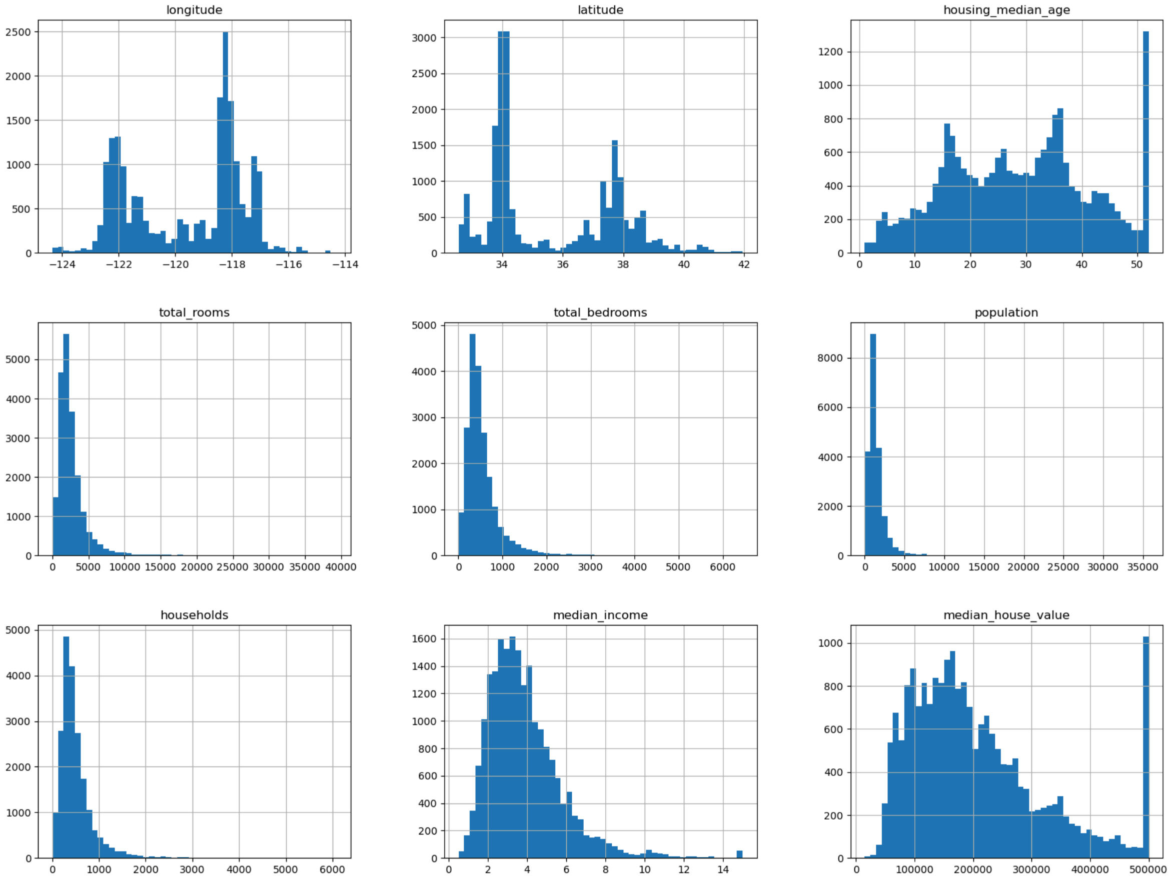
```
housing.describe()
```

Out[8]:

	longitude	latitude	housing_median_age	total_rooms	total_be
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.
mean	-119.569704	35.631861	28.639486	2635.763081	537
std	2.003532	2.135952	12.585558	2181.615252	421
min	-124.350000	32.540000	1.000000	2.000000	1.
25%	-121.800000	33.930000	18.000000	1447.750000	296.
50%	-118.490000	34.260000	29.000000	2127.000000	435.
75%	-118.010000	37.710000	37.000000	3148.000000	647.
max	-114.310000	41.950000	52.000000	39320.000000	6445.

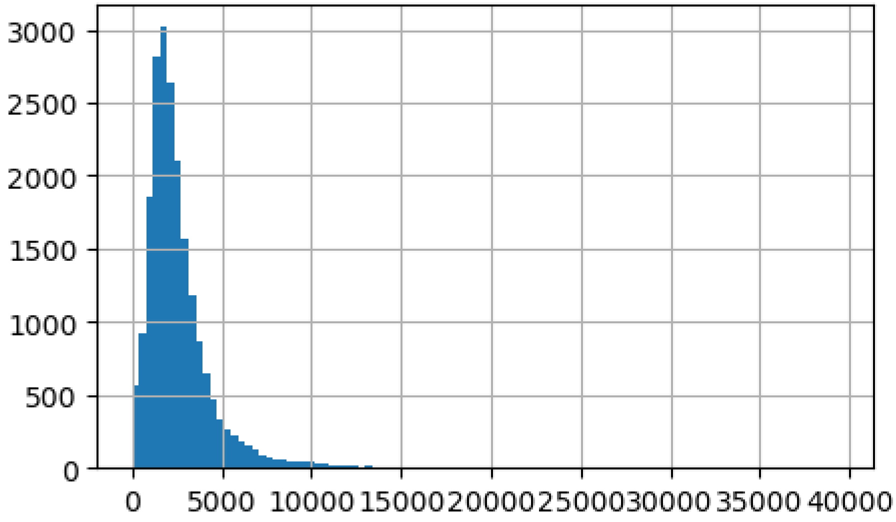
In [9]:

```
import matplotlib.pyplot as plt
housing.hist(bins=50,figsize=(20,15))
plt.show()
```



In [10]:

```
housing["total_rooms"].hist(bins=100,figsize=(5,3))
plt.show()
```



In [11]:

```
import numpy as np
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

In [12]:

```
train_set, test_set = split_train_test(housing, 0.2)
```

In [13]:

```
print(len(train_set), "train +", len(test_set), "test")
```

```
16512 train + 4128 test
```

In [14]:
```
housing["income_cat"] = np.ceil(housing["median_income"] / 1.5)
housing["income_cat"] = housing["income_cat"].where(housing["income_cat"]
```

In [15]:
```
housing.head()
```

Out[15]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-122.23	37.88	41.0	880.0	129.0	32
1	-122.22	37.86	21.0	7099.0	1106.0	240
2	-122.24	37.85	52.0	1467.0	190.0	49
3	-122.25	37.85	52.0	1274.0	235.0	55
4	-122.25	37.85	52.0	1627.0	280.0	56

In [16]:
```
from sklearn.model_selection import StratifiedShuffleSplit
```

In [17]:
```
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42
for train_index, test_index in split.split(housing, housing["income_cat"]
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

In [18]:
```
 housing["income_cat"].value_counts() / len(housing)
```

Out[18]:
```
income_cat
3.0    0.350581
2.0    0.318847
4.0    0.176308
5.0    0.114438
1.0    0.039826
Name: count, dtype: float64
```

In [19]:
```
for set in (strat_train_set, strat_test_set):
    set.drop(["income_cat"], axis=1, inplace=True)
```

In [20]:
```
housing.head()
```

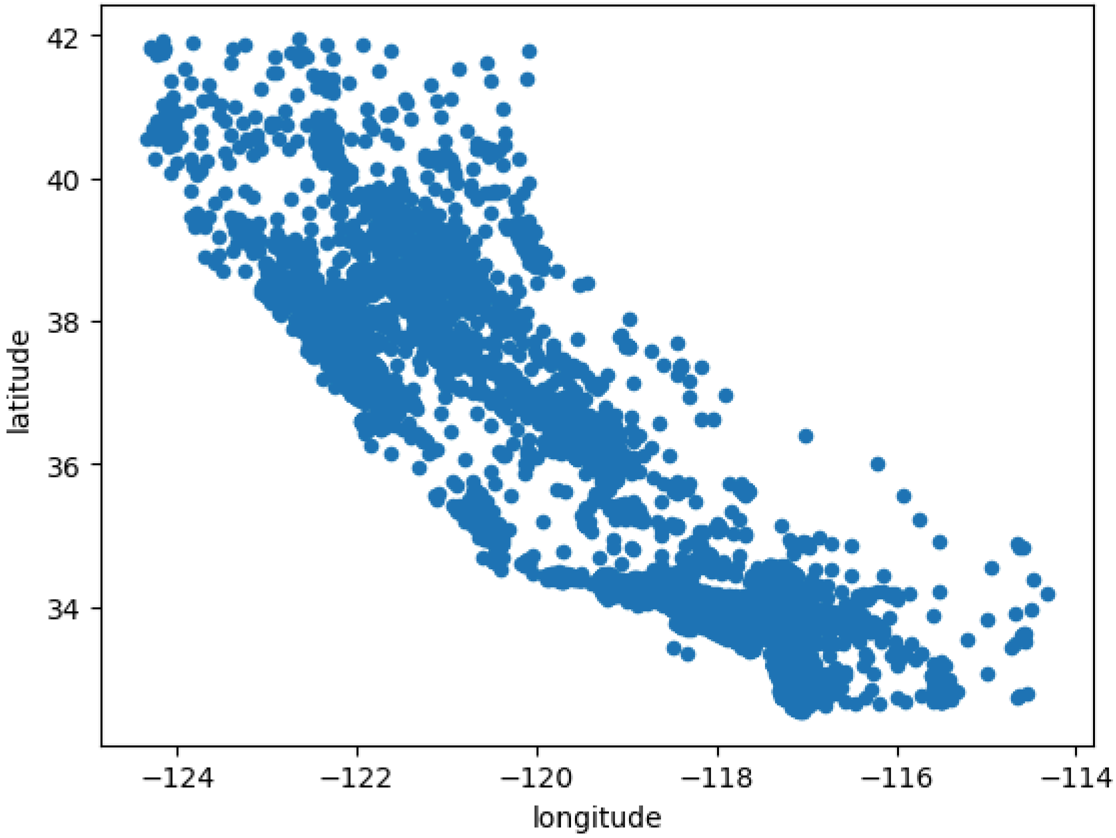
Out[20]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-122.23	37.88	41.0	880.0	129.0	32
1	-122.22	37.86	21.0	7099.0	1106.0	240
2	-122.24	37.85	52.0	1467.0	190.0	49
3	-122.25	37.85	52.0	1274.0	235.0	55
4	-122.25	37.85	52.0	1627.0	280.0	56

In [21]:
```
housing1 = strat_train_set.copy()
```

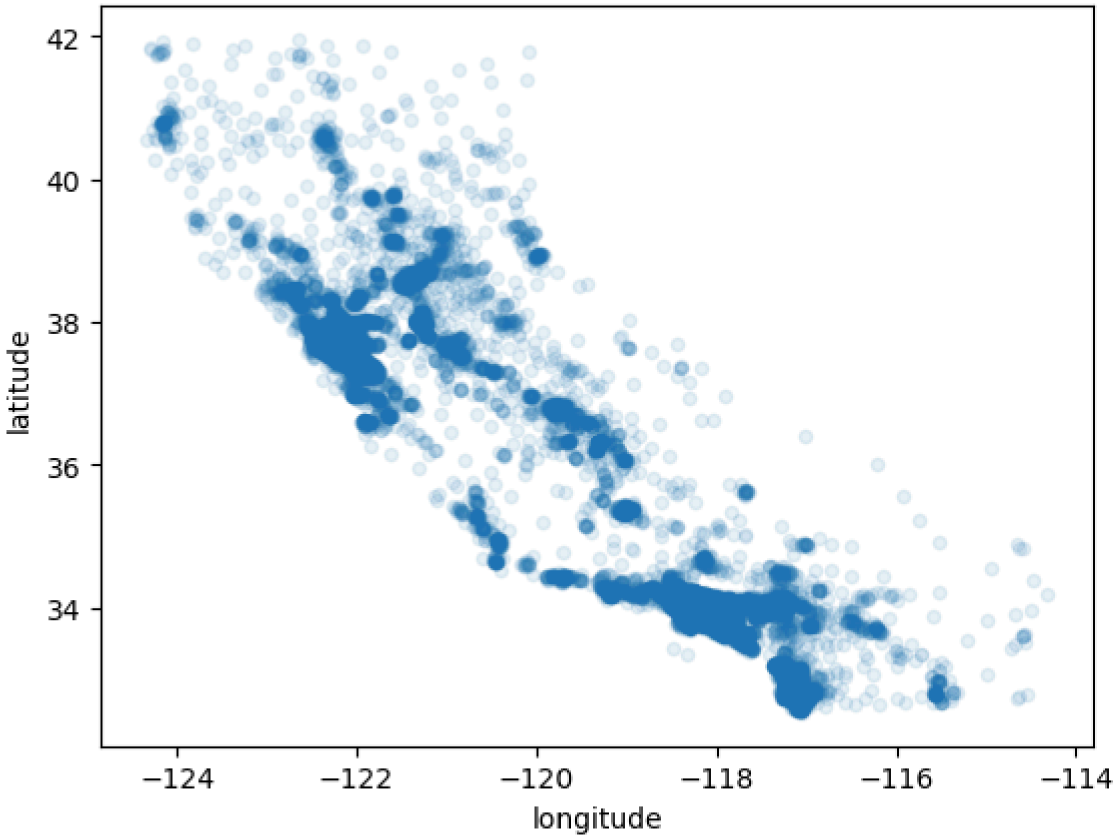
In [22]:
```
housing1.plot(kind="scatter", x="longitude", y="latitude")
```

Out[22]:
```
<Axes: xlabel='longitude', ylabel='latitude'>
```



In [23]:
```
housing1.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

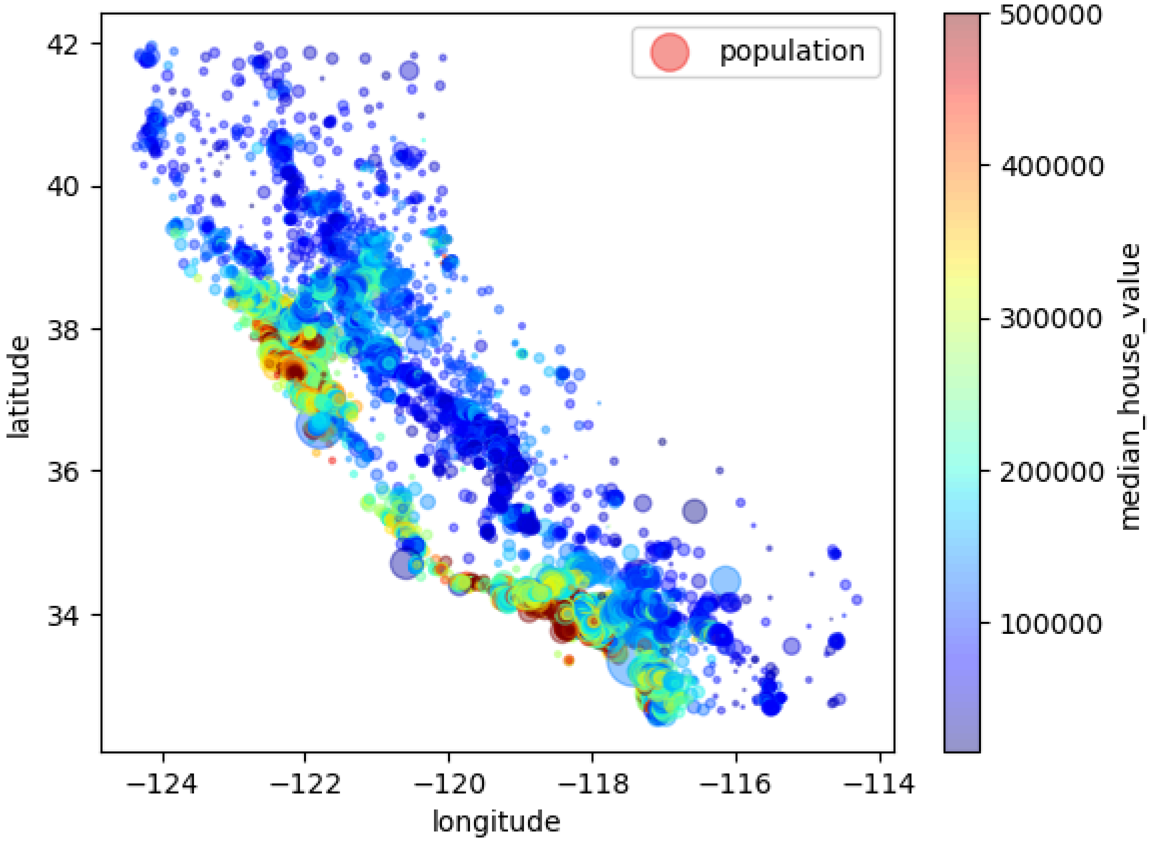
Out[23]:
```
<Axes: xlabel='longitude', ylabel='latitude'>
```



In [24]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
s=housing["population"]/100, label="population",
c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
)
plt.legend()
```

Out[24]:

```
<matplotlib.legend.Legend at 0x160e73470>
```



In [25]:

```
housing1.head()
```

Out[25]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	po
12655	-121.46	38.52	29.0	3873.0	797.0	
15502	-117.23	33.09	7.0	5320.0	855.0	
2908	-119.04	35.37	44.0	1618.0	310.0	
14053	-117.13	32.75	24.0	1877.0	519.0	
20496	-118.70	34.28	27.0	3536.0	646.0	

In [26]:

```
housing.head()
```

Out[26]:

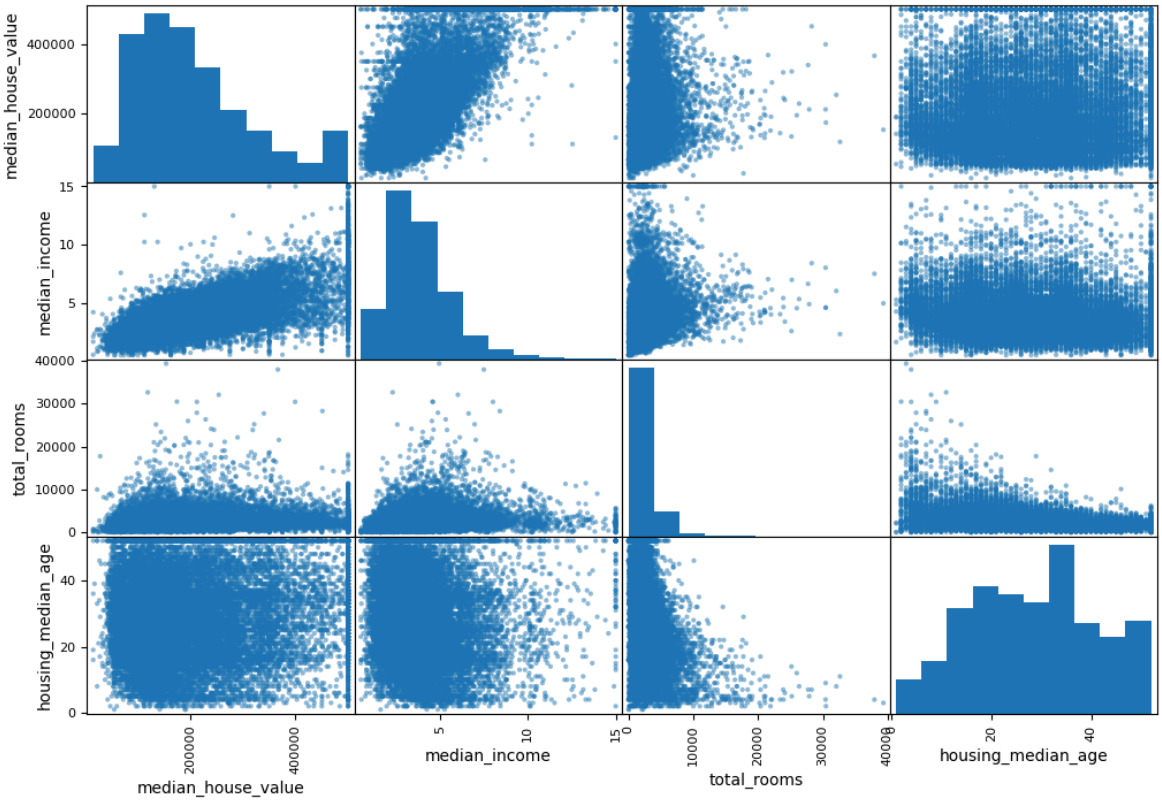
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-122.23	37.88	41.0	880.0	129.0	32
1	-122.22	37.86	21.0	7099.0	1106.0	240
2	-122.24	37.85	52.0	1467.0	190.0	49
3	-122.25	37.85	52.0	1274.0	235.0	55
4	-122.25	37.85	52.0	1627.0	280.0	56

In [27]:

```
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms","housi
scatter_matrix(housing[attributes], figsize=(12, 8))
```

Out[27]:

```
array([[<Axes: xlabel='median_house_value', ylabel='median_house_valu
e'>,
        <Axes: xlabel='median_income', ylabel='median_house_value'>,
        <Axes: xlabel='total_rooms', ylabel='median_house_value'>,
        <Axes: xlabel='housing_median_age', ylabel='median_house_valu
e'>],
       [<Axes: xlabel='median_house_value', ylabel='median_income'>,
        <Axes: xlabel='median_income', ylabel='median_income'>,
        <Axes: xlabel='total_rooms', ylabel='median_income'>,
        <Axes: xlabel='housing_median_age', ylabel='median_income'>],
       [<Axes: xlabel='median_house_value', ylabel='total_rooms'>,
        <Axes: xlabel='median_income', ylabel='total_rooms'>,
        <Axes: xlabel='total_rooms', ylabel='total_rooms'>,
        <Axes: xlabel='housing_median_age', ylabel='total_rooms'>],
       [<Axes: xlabel='median_house_value', ylabel='housing_median_ag
e'>,
        <Axes: xlabel='median_income', ylabel='housing_median_age'>,
        <Axes: xlabel='total_rooms', ylabel='housing_median_age'>,
        <Axes: xlabel='housing_median_age', ylabel='housing_median_ag
e'>]],
      dtype=object)
```

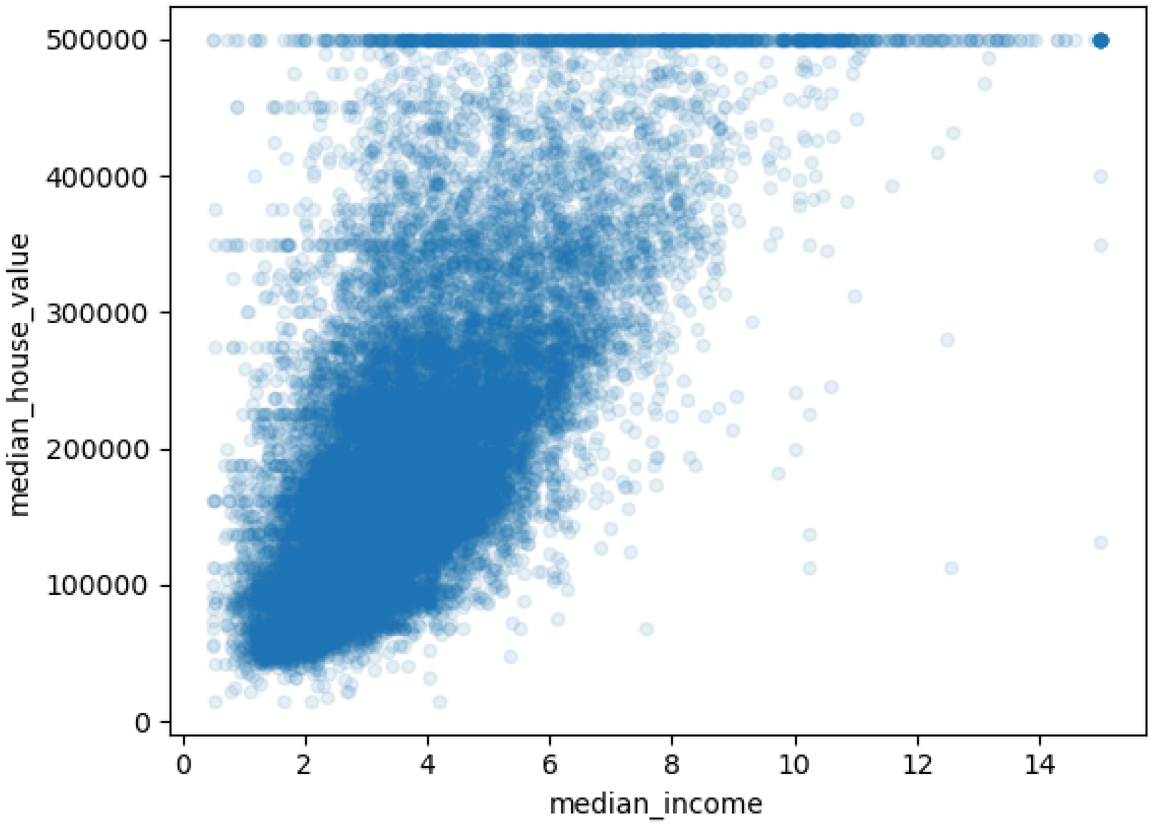


In [28]:

```
housing.plot(kind="scatter", x="median_income", y="median_house_value",al
```

Out[28]:

```
<Axes: xlabel='median_income', ylabel='median_house_value'>
```



In [29]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["househol
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_r
housing["population_per_household"]=housing["population"]/housing["househ
```

In [30]:
```
housing.head()
```

Out[30]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-122.23	37.88	41.0	880.0	129.0	32
1	-122.22	37.86	21.0	7099.0	1106.0	240
2	-122.24	37.85	52.0	1467.0	190.0	49
3	-122.25	37.85	52.0	1274.0	235.0	55
4	-122.25	37.85	52.0	1627.0	280.0	56

In [31]:
```
housing2=strat_train_set.drop("median_house_value",axis = 1)
housing_labels = strat_train_set["median_house_value"].copy()
```

In [32]:
```
housing2.head()
```

Out[32]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	po
12655	-121.46	38.52	29.0	3873.0	797.0	
15502	-117.23	33.09	7.0	5320.0	855.0	
2908	-119.04	35.37	44.0	1618.0	310.0	
14053	-117.13	32.75	24.0	1877.0	519.0	
20496	-118.70	34.28	27.0	3536.0	646.0	

In [33]:

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
housing_num = housing2.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
```

Out[33]:

```
▾ SimpleImputer
SimpleImputer(strategy='median')
```

In [34]:

```
imputer.statistics_
```

Out[34]:

```
array([-118.51    ,   34.26    ,   29.     , 2119.     ,  433.     ,
        1164.     ,  408.     ,    3.54155])
```

In [35]:

```
housing_num.median().values
```

Out[35]:

```
array([-118.51    ,   34.26    ,   29.     , 2119.     ,  433.     ,
        1164.     ,  408.     ,    3.54155])
```

In [36]:

```
X = imputer.transform(housing_num)
```

In [37]:

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns)
```

In [38]:

```
housing_tr.head()
```

Out[38]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-121.46	38.52	29.0	3873.0	797.0	223
1	-117.23	33.09	7.0	5320.0	855.0	201
2	-119.04	35.37	44.0	1618.0	310.0	66
3	-117.13	32.75	24.0	1877.0	519.0	89
4	-118.70	34.28	27.0	3536.0	646.0	183

In [86]:

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
housing_cat = housing2["ocean_proximity"]
housing_cat_encoded = encoder.fit_transform(housing_cat)
```

In [88]:

```
housing_cat_encoded
```

Out[88]:

```
array([1, 4, 1, ..., 0, 0, 1])
```

In [90]:

```
print(encoder.classes_)
```

```
['<1H OCEAN' 'INLAND' 'ISLAND' 'NEAR BAY' 'NEAR OCEAN']
```

In [92]:

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1
housing_cat_1hot
```

Out[92]:

```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
        with 16512 stored elements in Compressed Sparse Row format>
```

In [94]:

```
housing_cat_1hot.toarray()
```

Out[94]:

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

In [96]:

```
from sklearn.preprocessing import LabelBinarizer
encoder = LabelBinarizer()
housing_cat_1hot = encoder.fit_transform(housing_cat)
housing_cat_1hot
```

Out[96]:

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

In [98]:

```
from sklearn.base import BaseEstimator, TransformerMixin
rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True):
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
        population_per_household = X[:, population_ix] / X[:, household_i
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household
        else:
            return np.c_[X, rooms_per_household, population_per_household
attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing2.values)
```

In [102…

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])
housing_num_tr = num_pipeline.fit_transform(housing_num)
```

In [108…

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]
```

```
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore')),
])

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", cat_pipeline, cat_attribs),
])

housing_prepared = full_pipeline.fit_transform(housing2)
```

In [110…

```
housing_prepared
```

Out[110…

```
array([[-0.94135046,  1.34743822,  0.02756357, ...,  0.        ,
         0.        ,  0.        ],
       [ 1.17178212, -1.19243966, -1.72201763, ...,  0.        ,
         0.        ,  1.        ],
       [ 0.26758118, -0.1259716 ,  1.22045984, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [-1.5707942 ,  1.31001828,  1.53856552, ...,  0.        ,
         0.        ,  0.        ],
       [-1.56080303,  1.2492109 , -1.1653327 , ...,  0.        ,
         0.        ,  0.        ],
       [-1.28105026,  2.02567448, -0.13148926, ...,  0.        ,
         0.        ,  0.        ]])
```

In [112…

```
housing_prepared.shape
```

Out[112…

```
(16512, 16)
```

In [114…

```
from sklearn.base import BaseEstimator, TransformerMixin
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

In [116…

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
some_data = housing2.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)
print("Predictions:\t", lin_reg.predict(some_data_prepared))
```

```
Predictions:     [ 86208. 304704. 153536. 185728. 244416.]
```

In [118…

```
print("Labels:\t\t", list(some_labels))
```

```
Labels:          [72100.0, 279600.0, 82700.0, 112500.0, 238300.0]
```

In [120…

```
from sklearn.metrics import mean_squared_error
housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[120…

```
68633.40810776998
```

In [122…

```
from sklearn.tree import DecisionTreeRegressor
tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[122…

```
0.0
```

In [124…

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,scori
rmse_scores = np.sqrt(-scores)
```

In [126…

```
rmse_scores
```

Out[126…

```
array([73036.77479653, 71162.50050245, 67631.65395197, 71349.78951703,
       68170.49527976, 78024.8224294 , 71221.48531253, 72697.72723185,
       66738.43077005, 71369.36419196])
```

In [130…

```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
display_scores(rmse_scores)
```

```
Scores: [73036.77479653 71162.50050245 67631.65395197 71349.78951703
 68170.49527976 78024.8224294  71221.48531253 72697.72723185
 66738.43077005 71369.36419196]
Mean: 71140.30439835272
Standard deviation: 3066.416073861671
```

In [132…

```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,sc
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
Scores: [71800.38078269 64114.99166359 67844.95431254 68635.19072082
 66801.98038821 72531.04505346 73992.85834976 68824.54092094
 66474.60750419 70143.79750458]
Mean: 69116.4347200802
Standard deviation: 2880.6588594759014
```

In [138…

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared, housing_labels)

housing_predictions = forest_reg.predict(housing_prepared)
```

```
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)

forest_rmse
```

Out[138…

```
18872.0124301936
```

In [140…

```
forest_rmse_scores = np.sqrt(-cross_val_score(forest_reg,housing_prepared

display_scores(forest_rmse_scores)
```

```
Scores: [51985.82497671 49217.83935411 46823.56787575 51776.52196355
 47398.41564375 52220.58843745 52138.19825663 50096.45187055
 48476.73132457 53807.30656357]
Mean: 50394.14462666457
Standard deviation: 2221.3488519264606
```

In [142…

```
from sklearn.model_selection import GridSearchCV
param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3
]
forest_reg = RandomForestRegressor()
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,scoring='neg_mean
grid_search.fit(housing_prepared, housing_labels)
```

Out[142…

```
▸ GridSearchCV
▸ best_estimator_: RandomForestRegressor
    ▸ RandomForestRegressor
```

In [144…

```
grid_search.best_params_
```

Out[144…

```
{'max_features': 6, 'n_estimators': 30}
```

In [146…

```
grid_search.best_estimator_
```

Out[146…

```
▾ RandomForestRegressor
RandomForestRegressor(max_features=6, n_estimators=30)
```

In [150…

```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
64073.18746458702 {'max_features': 2, 'n_estimators': 3}
55648.52461660227 {'max_features': 2, 'n_estimators': 10}
52712.25739188647 {'max_features': 2, 'n_estimators': 30}
59782.389696785925 {'max_features': 4, 'n_estimators': 3}
53145.331899038676 {'max_features': 4, 'n_estimators': 10}
50670.05777146122 {'max_features': 4, 'n_estimators': 30}
59006.48216827552 {'max_features': 6, 'n_estimators': 3}
52120.468713441966 {'max_features': 6, 'n_estimators': 10}
49980.4746877061 {'max_features': 6, 'n_estimators': 30}
58951.09657082649 {'max_features': 8, 'n_estimators': 3}
52171.651235856676 {'max_features': 8, 'n_estimators': 10}
50150.42381547754 {'max_features': 8, 'n_estimators': 30}
62233.020880427095 {'bootstrap': False, 'max_features': 2, 'n_estimators':
3}
54336.966822250375 {'bootstrap': False, 'max_features': 2, 'n_estimators':
10}
59722.058306681414 {'bootstrap': False, 'max_features': 3, 'n_estimators':
3}
52527.419275604065 {'bootstrap': False, 'max_features': 3, 'n_estimators':
10}
58141.62386462325 {'bootstrap': False, 'max_features': 4, 'n_estimators':
3}
51462.05091090205 {'bootstrap': False, 'max_features': 4, 'n_estimators':
10}
```

In [152…

```
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

Out[152…

```
array([7.02869080e-02, 5.88110949e-02, 4.47377380e-02, 1.89453748e-02,
       1.74848339e-02, 1.97364709e-02, 1.59457115e-02, 3.21030056e-01,
       6.06899401e-02, 1.06598997e-01, 8.91566676e-02, 9.17006583e-03,
       1.59248020e-01, 9.55352481e-05, 3.02551824e-03, 5.03706828e-03])
```

In [154…

```
extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
cat_one_hot_attribs = list(encoder.classes_)
attributes = num_attribs + extra_attribs + cat_one_hot_attribs
sorted(zip(feature_importances, attributes), reverse=True)
```

Out[154…

```
[(0.32103005551696234, 'median_income'),
 (0.15924801993991142, 'INLAND'),
 (0.10659899707798415, 'pop_per_hhold'),
 (0.08915666759533054, 'bedrooms_per_room'),
 (0.07028690796898827, 'longitude'),
 (0.06068994010859933, 'rooms_per_hhold'),
 (0.05881109492548774, 'latitude'),
 (0.04473773800549152, 'housing_median_age'),
 (0.01973647094052662, 'population'),
 (0.018945374839328212, 'total_rooms'),
 (0.01748483394303946, 'total_bedrooms'),
 (0.015945711539215004, 'households'),
 (0.009170065834496555, '<1H OCEAN'),
 (0.005037068281272642, 'NEAR OCEAN'),
 (0.0030255182352539782, 'NEAR BAY'),
 (9.553524811216867e-05, 'ISLAND')]
```

In [156…

```
final_model = grid_search.best_estimator_
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()
X_test_prepared = full_pipeline.transform(X_test)
```

In [158…

```
final_predictions = final_model.predict(X_test_prepared)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [160…

```
final_rmse
```

Out[160…

```
48272.361023717815
```

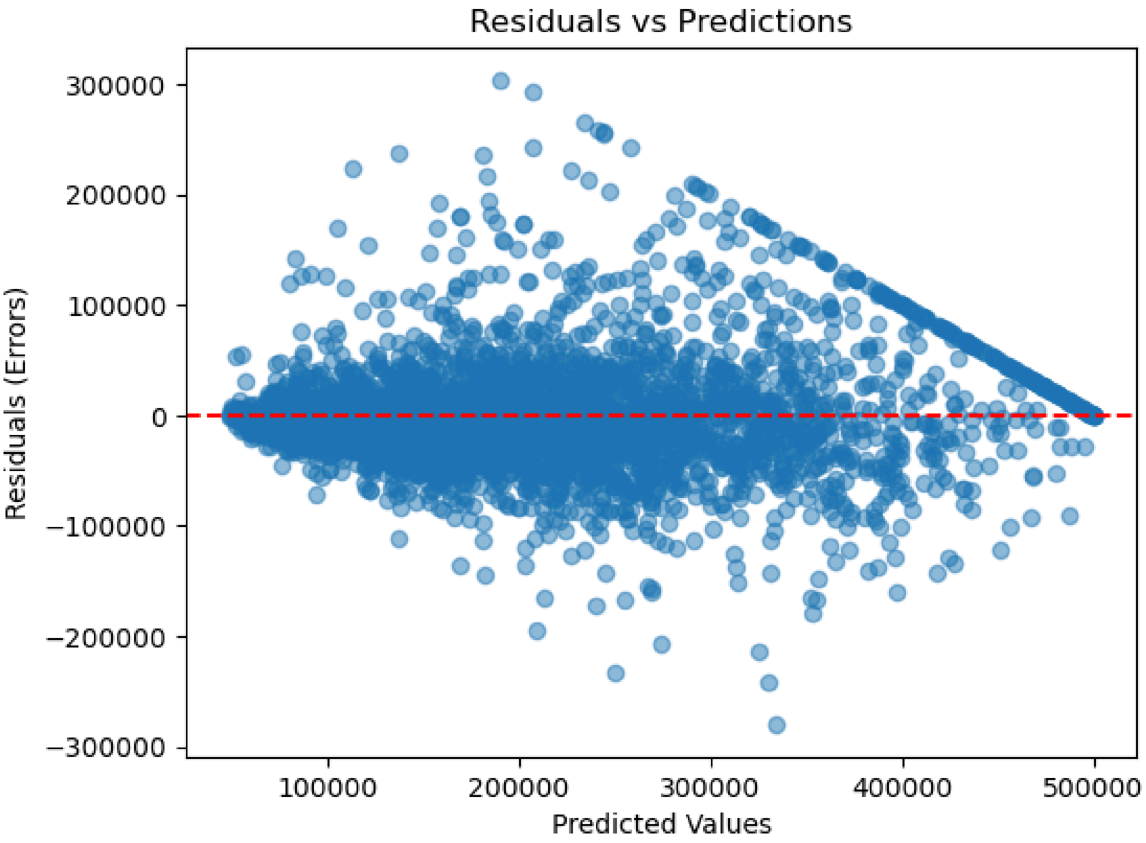
In [162…

```
print("Best CV RMSE:", np.sqrt(-grid_search.best_score_))
```

```
Best CV RMSE: 49980.4746877061
```

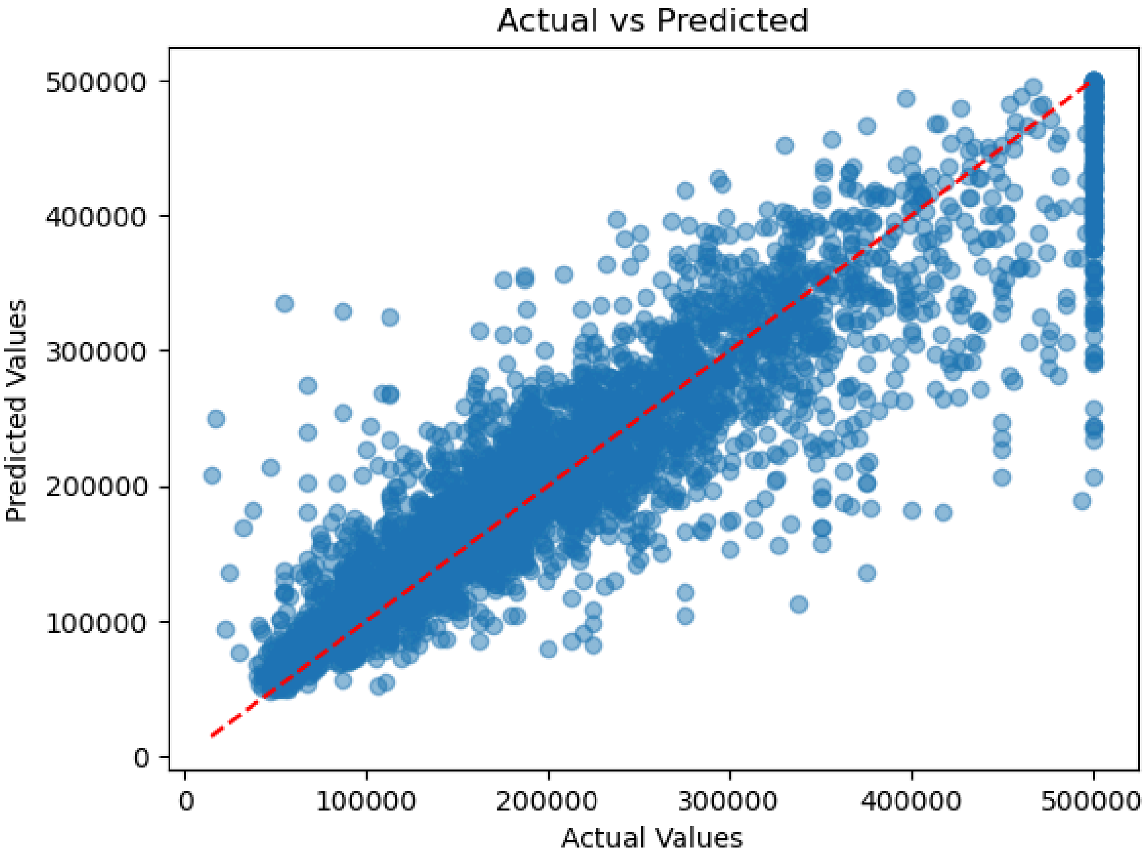
In [164…

```
residuals = y_test - final_predictions
plt.scatter(final_predictions, residuals, alpha=0.5)
plt.axhline(y=0, color="red", linestyle="--")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals (Errors)")
plt.title("Residuals vs Predictions")
plt.show()
```



In [166…

```
plt.scatter(y_test, final_predictions, alpha=0.5)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted")
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()], "r--")
plt.show()
```


Actual vs Predicted
Predicted Values
500000
400000
300000
200000
100000
0
Actual Values
0
100000
200000
300000
400000
500000