In [1]:
```python
import pandas as pd
import numpy as np
```

In [5]:
```python
path = '/Users/User/VSCodeFiles/Projects/CaliHousing/archive'
df = pd.read_csv(path+'/housing.csv')
```

In [8]:
```python
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
```

In [18]:
```python
null = df.total_bedrooms.isnull().sum()
print(null)
```

```
207
```

207 null vs 20,000 rows. Will just remove

In [24]:
```python
df1 = df.dropna().reset_index(drop=True)
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20433 entries, 0 to 20432
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  float64
 3   total_rooms         20433 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20433 non-null  float64
 6   households          20433 non-null  float64
 7   median_income       20433 non-null  float64
 8   median_house_value  20433 non-null  float64
 9   ocean_proximity     20433 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [28]:
```python
duplicates = df1.duplicated()
print(duplicates.sum())
```

```
0
```

No duplicates

In [38]:
```python
## Need to one hot encode categorical data
pd.unique(df1['ocean_proximity'])
```

Out[38]:
```
array(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'],
      dtype=object)
```

In [36]:
```python
df_dummies = pd.get_dummies(df1,columns=['ocean_proximity'])
df_dummies
```

Out[36]:

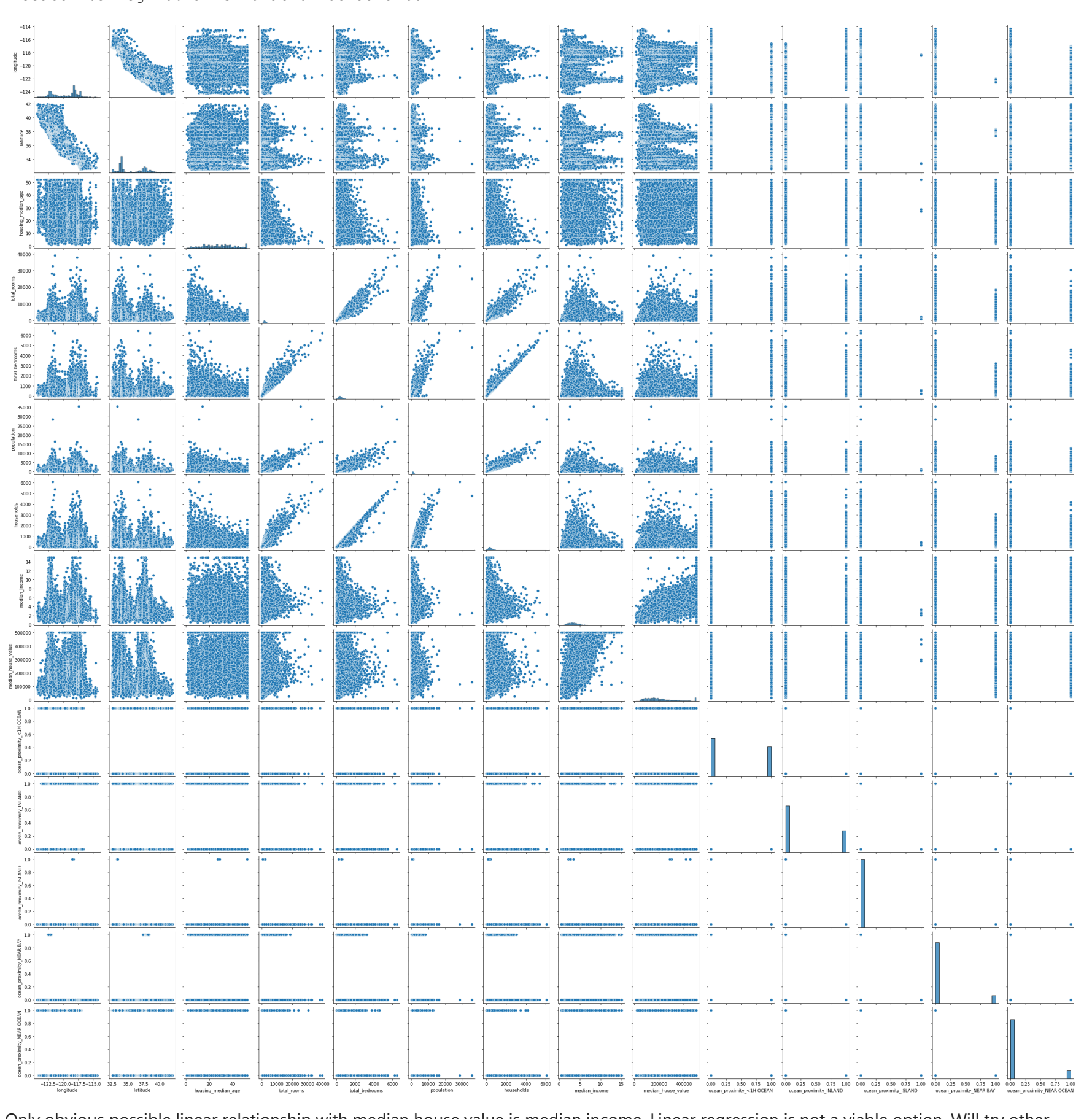
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0
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
...	...	...	...	...	...	...	...	...	...
20428	-121.09	39.48	25.0	1665.0	374.0	845.0	330.0	1.5603	78100.0
20429	-121.21	39.49	18.0	697.0	150.0	356.0	114.0	2.5568	77100.0
20430	-121.22	39.43	17.0	2254.0	485.0	1007.0	433.0	1.7000	92300.0
20431	-121.32	39.43	18.0	1860.0	409.0	741.0	349.0	1.8672	84700.0
20432	-121.24	39.37	16.0	2785.0	616.0	1387.0	530.0	2.3886	89400.0

20433 rows × 14 columns

In [37]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
```

In [39]:
```python
sns.pairplot(df_dummies)
```

Out[39]: <seaborn.axisgrid.PairGrid at 0x1d8f9a292b0>



Only obvious possible linear relationship with median house value is median income. Linear regression is not a viable option. Will try other classifiers. Going to use random forest classifiers, no need to preprocess data

In [43]:
```python
Y = df_dummies['median_house_value']
X = df_dummies.drop(labels='median_house_value', axis = 1)
```

In [47]:
```python
## Splitting up data into training, validation, and testing data. 60/20/20
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.4)
```

In [52]:
```python
X_val, X_test, y_val, y_test =  train_test_split(X_test, y_test, test_size=0.5)
```

In [61]:
```python
## First model will be random forest regressor
## First use gridsearch to find best parameters

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import max_error, r2_score
```

In [63]:
```python
params_rf = {
    'n_estimators':[50,75,125],
    'max_depth':[3,6,None],
    'min_samples_split':[2,3],
    'max_features':[4,6]
    }

scoring = ('max_error', 'r2')
```

In [65]:
```python
rf = RandomForestRegressor(random_state=5)
rf_cv = GridSearchCV(rf, param_grid=params_rf, cv = 5, scoring=scoring,refit='r2')
```

In [69]:
```python
rf_cv.fit(X_train, y_train)
```

Out[69]:
```
GridSearchCV
estimator: RandomForestRegressor
RandomForestRegressor
```

In [70]:
```python
rf_cv.best_estimator_
```

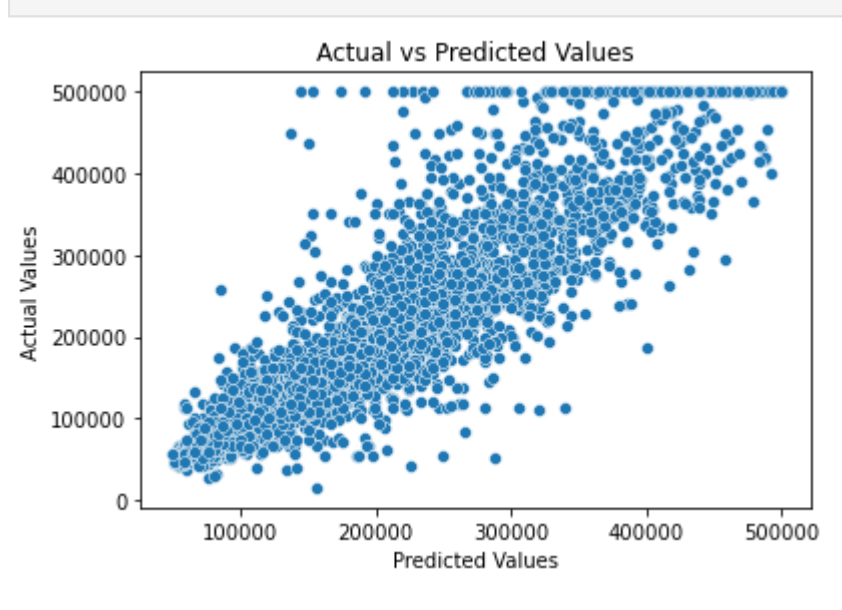
Out[70]:
```
RandomForestRegressor
RandomForestRegressor(max_features=6, n_estimators=125, random_state=5)
```

In [71]:
```python
rf_cv.best_score_
```

Out[71]: 0.809573320643534

In [72]:
```python
y_pred = rf_cv.predict(X_val)
```

In [78]:
```python
sns.scatterplot(x= y_pred, y=y_val)
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()
```



In [87]:
```python
r2 = r2_score(y_val,y_pred)
```

In [121...
```python
validation_table = pd.DataFrame()
```

In [122...
```python
Random_forest_regressor = {'Model':'RandomForestRegressor','r2': r2}

validation_table = validation_table.append(Random_forest_regressor, ignore_index=True)
```

In [96]:
```python
validation_table
```

Out[96]:

	Model	r2
0	RandomForestRegressor	0.810541

Now trying svm regressor, first need to normalize data

In [113...
```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_normal = scaler.fit_transform(X_train)

X_val_normal = scaler.fit_transform(X_val)
```

In [102...
```python
from sklearn.svm import SVR

svm_params = {
    'kernel':['linear','poly','sigmoid'],
    'degree':[3,4,5],
}
svm = SVR()
svm_cv = GridSearchCV(svm,param_grid=svm_params,scoring=scoring,cv=5,refit='r2')
```

In [104...
```python
svm_cv.fit(X_train_normal,y_train)
```

Out[104...
```
GridSearchCV
estimator: SVR
SVR
```

In [106...
```python
svm_cv.best_params_
```

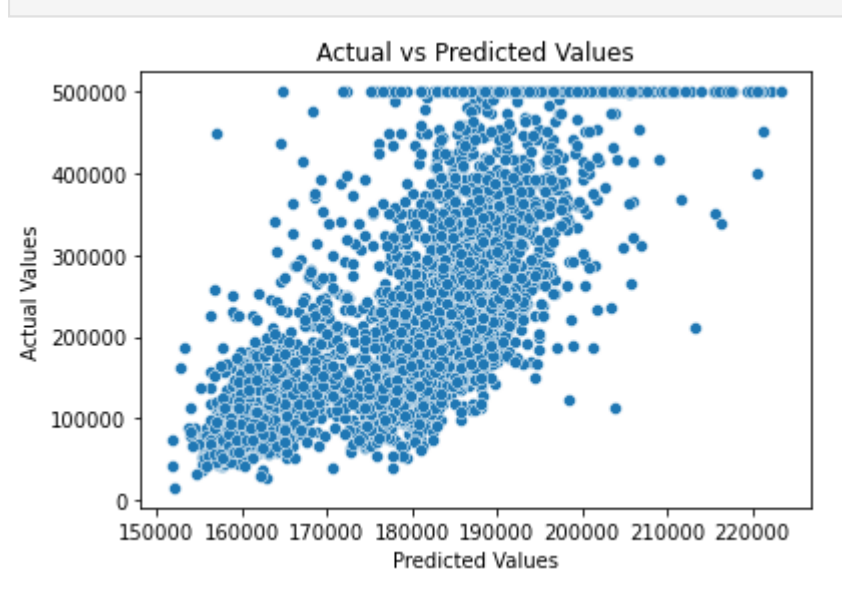
Out[106... {'degree': 3, 'kernel': 'linear'}

In [107...
```python
svm_cv.best_score_
```

Out[107... 0.05529052338590519

In [114...
```python
y_pred_svm = svm_cv.predict(X_val_normal)
```

In [115...
```python
sns.scatterplot(x= y_pred_svm, y=y_val)
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()
```



In [123...
```python
r2_svm = r2_score(y_val,y_pred_svm)
SVM_regressor = {'Model':'SVMRegressor','r2': r2_svm}

validation_table = validation_table.append(SVM_regressor, ignore_index=True)
```

In [124...
```python
validation_table
```

Out[124...

	Model	r2
0	RandomForestRegressor	0.810541
1	SVMRegressor	0.076776

SVM Regressor not effective

In [126...
```python
from sklearn.ensemble import GradientBoostingRegressor

gb_params = {
    'learning_rate':[0.1,0.3,0.5],
    'n_estimators':[75,100,200],
    'subsample':[0.5,0.75,1.0],
    'min_samples_split':[2,3,5],
}
gb = GradientBoostingRegressor()
gb_cv = GridSearchCV(gb,param_grid=gb_params,scoring=scoring,cv=5,refit='r2')
```

In [127...
```python
gb_cv.fit(X_train,y_train)
```

Out[127...
```
GridSearchCV
estimator: GradientBoostingRegressor
GradientBoostingRegressor
```

In [128...
```python
gb_cv.best_params_
```

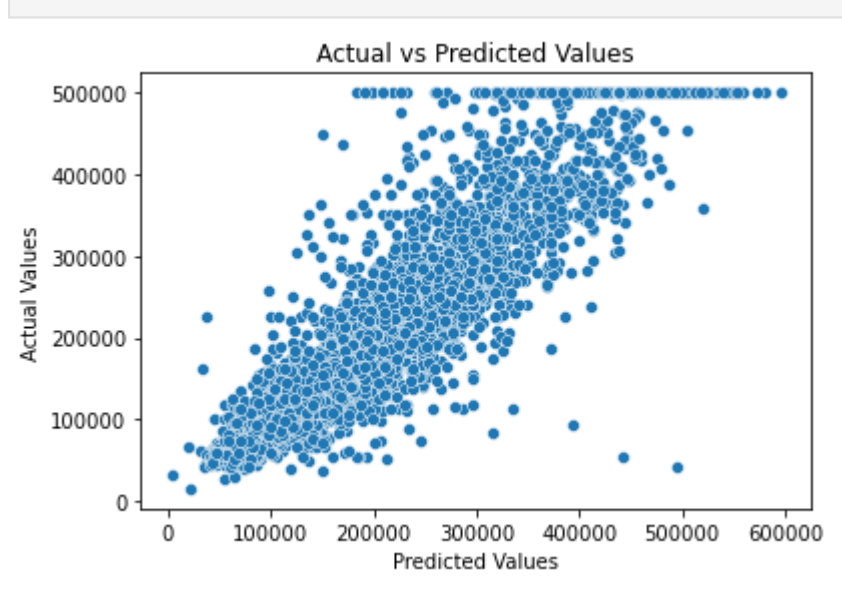
Out[128...
```
{'learning_rate': 0.3,
 'min_samples_split': 3,
 'n_estimators': 200,
 'subsample': 1.0}
```

In [129...
```python
gb_cv.best_score_
```

Out[129... 0.8099260579842472

In [131...
```python
y_pred_gb = gb_cv.predict(X_val)
```

In [132...
```python
sns.scatterplot(x= y_pred_gb, y=y_val)
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()
```



In [133...
```python
r2_gb = r2_score(y_val,y_pred_gb)
gb_regressor = {'Model':'Gradien Boosting Regressor','r2': r2_gb}

validation_table = validation_table.append(gb_regressor, ignore_index=True)
```

In [134...
```python
validation_table
```

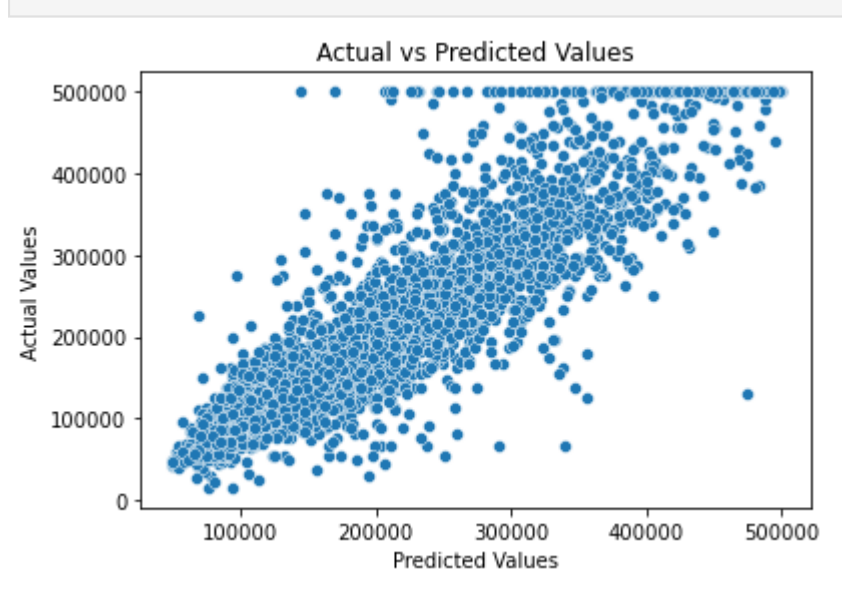
Out[134...

	Model	r2
0	RandomForestRegressor	0.810541
1	SVMRegressor	0.076776
2	Gradien Boosting Regressor	0.806554

Of the models tested, random forest regressor has the best r2 score. Will now test on testing data and create evaluation visualizations

In [135...
```python
y_pred_rf_test = rf_cv.predict(X_test)
```

In [136...
```python
sns.scatterplot(x= y_pred_rf_test, y=y_test)
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()
```



In [138...
```python
r2_rf_final = r2_score(y_test,y_pred_rf_test)
r2_rf_final
```

Out[138... 0.818056224147396

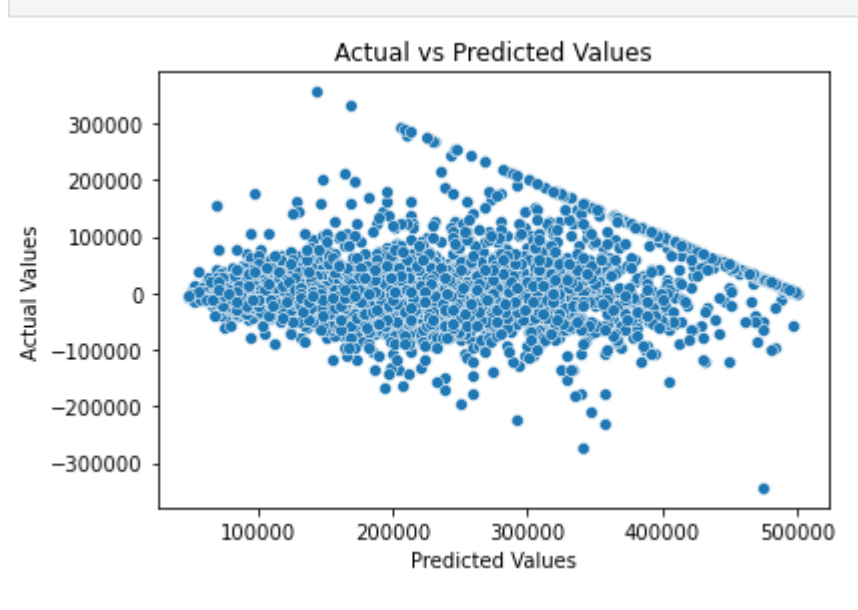
In [139...
```python
final_regressor = {'Model':'Random Forest on Test Data','r2': r2_rf_final}

validation_table = validation_table.append(final_regressor, ignore_index=True)
```

In [140...
```python
validation_table
```

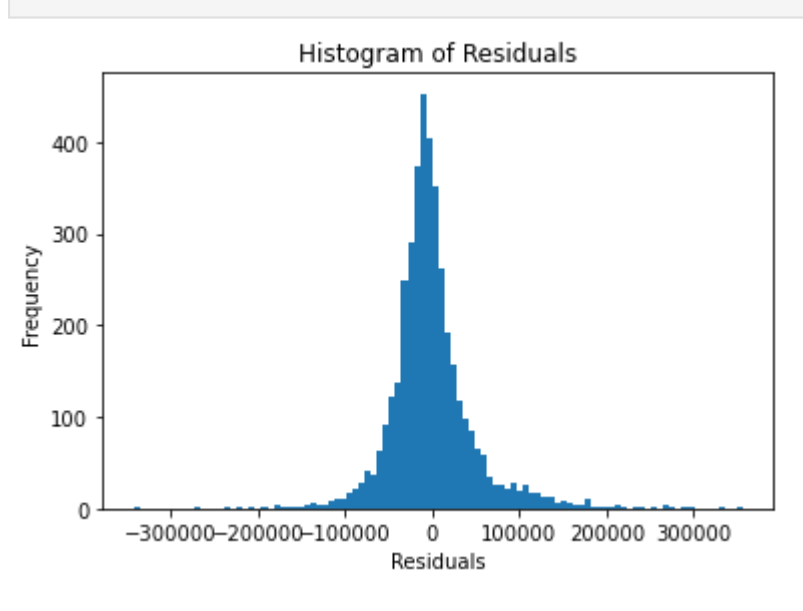
Out[140...

	Model	r2
0	RandomForestRegressor	0.810541
1	SVMRegressor	0.076776
2	Gradien Boosting Regressor	0.806554
3	Random Forest on Test Data	0.818056

In [142...
```python
residuals = y_test - y_pred_rf_test
sns.scatterplot(x= y_pred_rf_test, y=residuals)
plt.ylabel('Residuals')
plt.xlabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()
```

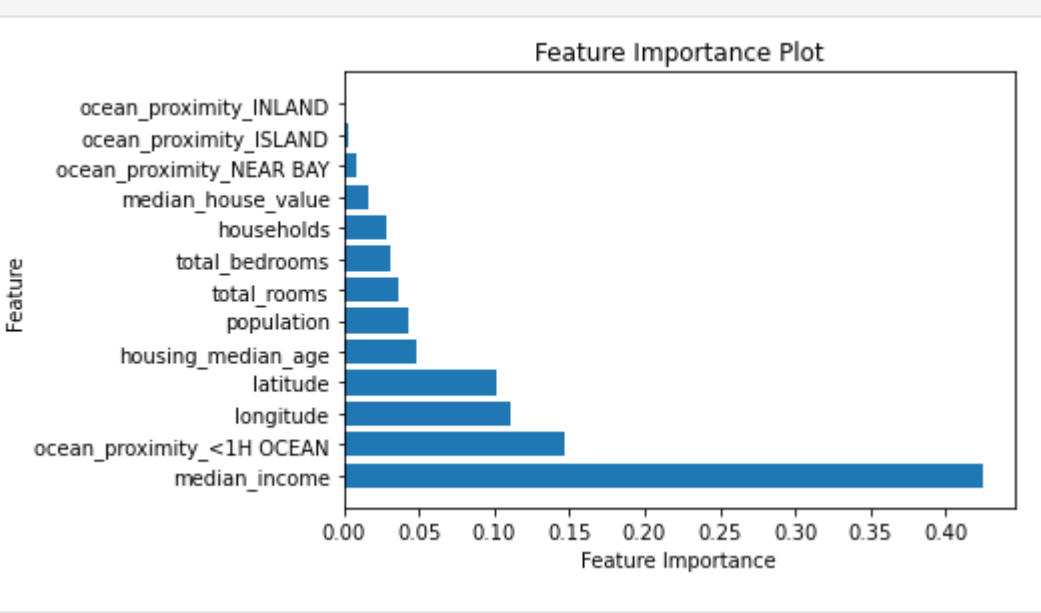


In [144...
```python
plt.hist(residuals, bins=100)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Histogram of Residuals')
plt.show()
```



In [150...
```python
best_model = rf_cv.best_estimator_
```

In [152...
```python
feature_importances = best_model.feature_importances_

feature_names = df_dummies.columns

sorted_indices = feature_importances.argsort()[::-1]
sorted_feature_importances = feature_importances[sorted_indices]
sorted_feature_names = feature_names[sorted_indices]

plt.barh(range(len(sorted_feature_importances)), sorted_feature_importances, align='center')
plt.yticks(range(len(sorted_feature_importances)), sorted_feature_names)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importance Plot')
plt.show()
```



In [154...
```python
from sklearn import tree

estimator = best_model.estimators_[0]

# Plot the decision tree
plt.figure(figsize=(12, 8))
tree.plot_tree(estimator, feature_names=feature_names, filled=True)
plt.show()
```