California Housing Dataset

The data pertains to the houses found in a given California district, it was gotten from Kaggle, and some summary stats about them based on the 1990 census data. Be warned the data aren't cleaned so there are some preprocessing steps required! The columns are as follows, their names are pretty self explanitory:

- longitude
- latitude
- housingmedianage
- total_rooms
- total_bedrooms
- population
- households
- median_income
- medianhousevalue
- ocean_proximity

Acknowledgements

This data was initially featured in the following paper: Pace, R. Kelley, and Ronald Barry. "Sparse spatial autoregressions." Statistics & Probability Letters 33.3 (1997): 291-297.

Main objective of Analysis

To perform Linear Regression Analysis predicting the median housing price and come up with a model that best fits the data and makes optimum prediction with reasonable accuracy

Procedures

- Explore the data
- Data Preprocessing and Preparation
- Trained models

In [2]:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:

```
housing = pd.read_csv('housing.csv')
housing.shape
display(housing.head())
housing.info()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean
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	

```
<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 [4]:

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

Out[4]:

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

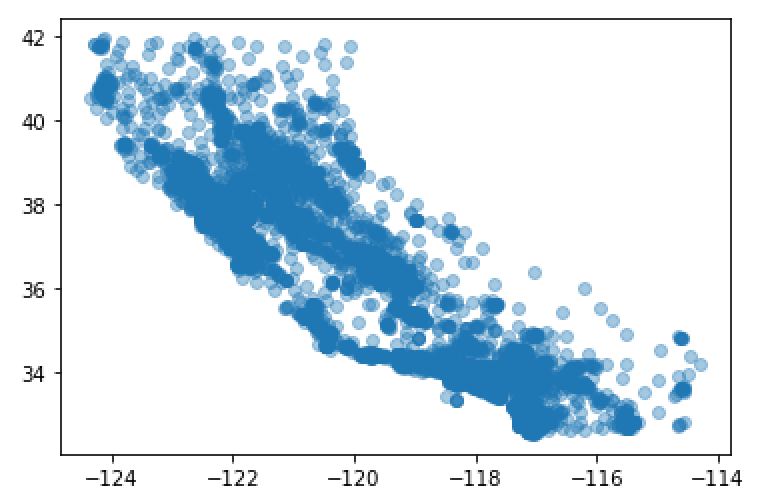
In [5]:

```
housing['total_bedrooms'] = housing['total_bedrooms'].fillna(housing.total_bedrooms.median())
housing.total_bedrooms.isnull().sum()
```

Out[5]: 0

In [6]:

```
plt.scatter(housing.longitude, housing.latitude, alpha=.4)
plt.show()
```



In [178…

```
# Preprocessing Data
y = housing['median_house_value']
X = housing.drop('median_house_value', axis=1)
numeric_cols = [col for col in X.columns if X[col].dtype == np.float64]
cat_cols = [col for col in X.columns if X[col].dtype == np.object]
X_numeric = X[numeric_cols]
X_cat = X[cat_cols]

# Scaling and OneHotEncoding
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()
# onehot = OneHotEncoder(handle_unknown='ignore')

X_num_tran = scaler.fit_transform(X_numeric)
X_num_tran = pd.DataFrame(X_num_tran, columns=numeric_cols)
# X_num_tran['location'] = X_num_tran['latitude'] / X_num_tran['longitude']
# if 'location' in X_num_tran.columns:
#     X_num_tran.drop(['latitude', 'longitude'], axis=1, inplace=True)

# numeric_cols
corr_df = X_num_tran.corr().abs()
mask = np.triu(np.ones_like(corr_df, dtype=bool))
tri_df = corr_df.mask(mask)
tri_df
to_drop = [c for c in tri_df.columns if any(tri_df[c] > 0.91)]
display(to_drop)
X_numeric_new = X_num_tran.drop('total_bedrooms', axis=1)

X_cat_tran = pd.get_dummies(X_cat, drop_first=True)


X_numeric_new.columns
```

```
['longitude', 'total_rooms', 'total_bedrooms']
```

Out[178…

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

In [193…

```
from sklearn.preprocessing import PolynomialFeatures

features = X_num_tran.columns
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X_num_tran)
X_med_house = pd.DataFrame(X_poly, columns=poly.get_feature_names(input_features=features))

# Join the Categorical and Numerical columns back
X_new = pd.concat([X_med_house, X_cat_tran], axis=1)

X_new.head()
```

Out[193…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	longitude^2	longitude latitude	...
0	-1.327835	1.052548	0.982143	-0.804819	-0.972476	-0.974429	-0.977033	2.344766	1.763146	-1.397611	...
1	-1.322844	1.043185	-0.607019	2.045890	1.357143	0.861439	1.669961	2.332238	1.749916	-1.379970	...
2	-1.332827	1.038503	1.856182	-0.535746	-0.827024	-0.820777	-0.843637	1.782699	1.776427	-1.384144	...
3	-1.337818	1.038503	1.856182	-0.624215	-0.719723	-0.766028	-0.733781	0.932968	1.789757	-1.389327	...
4	-1.337818	1.038503	1.856182	-0.462404	-0.612423	-0.759847	-0.629157	-0.012881	1.789757	-1.389327	...

5 rows × 48 columns

In [197…

```
# Mute the sklearn warning about regularization
import warnings
warnings.filterwarnings('ignore', module='sklearn')
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import RidgeCV, LassoCV, ElasticNetCV



def scores(model):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    print('Model: {};\n\tTraining score: {},\n\tTest score: {},\n\tRMSE: {}'.format(model.__class__.__name__,
                                                                                model.score(X_train, y_train),
                                                                                model.score(X_test, y_test),
                                                                                rmse))
def rmse(model):
    return np.sqrt(mean_squared_error(y_test, model.predict(X_test)))

X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=.2, random_state=42)
lr = LinearRegression()

alphas = [0.05, 0.01, 0.5, 0.1, 0.5, 10, 20, 30, 50, 80, 100, 200, 500, 1000, 0.5e4, 1e4, 1e5, 1e6]
alphas2 = [1e-5, 1e-4, 0.005, 0.001, 0.05, 0.01, 1, 100, 1000, 10000, 1e5]
l1_ratios = np.linspace(0.1, 0.9, 9)


ridgeCV = RidgeCV(alphas=alphas, cv=4)
LassoCV = LassoCV(alphas=alphas2, max_iter=400, cv=4, n_jobs=-1)
elasticNetCV = ElasticNetCV(alphas=alphas2, l1_ratio=l1_ratios, max_iter=400, n_jobs=-1)
# cv_results = cross_val_score(lr, X_train, y_train, cv=5)
# cv_results
models = (lr, ridgeCV, LassoCV, elasticNetCV)
for model in models:
    scores(model)
print(ridgeCV.alpha_, LassoCV.alpha_)
```

```
Model: LinearRegression;
	Training score: 0.7052881181807746,
	Test score: 0.6568633131509709,
	RMSE: 67055.90169483995
Model: RidgeCV;
	Training score: 0.6846182663495426,
	Test score: 0.6685367163469873,
	RMSE: 65905.42184495278
Model: LassoCV;
	Training score: 0.6730323463437291,
	Test score: 0.6484360153237769,
	RMSE: 67874.33991913799
Model: ElasticNetCV;
	Training score: 0.677905171876138,
	Test score: 0.6629106935887019,
	RMSE: 66462.38436817947
500.0 1000.0
```

Discussion of Results from Model Training and Evaluation

- Linear Regression model overfitted the data due to the fact that i chose a 2nd polynomial degree
- Compensation by Regularization is observed in both RidgeCV and LassoCV models
- Best model was RidgeCV at alpha value of 500 and RMSE of 65905

In [195…

```
# Models and RMSE
labels = ['Linear', 'Ridge', 'Lasso', 'ElasticNet']
rmse_vals = [rmse(model) for model in models]

rmse_df = pd.Series(rmse_vals, index=labels).to_frame()
rmse_df.rename(columns={0: 'RMSE'}, inplace=1)
rmse_df
```

Out[195…

	RMSE
Linear	67055.901695
Ridge	65905.421845
Lasso	67874.339919
ElasticNet	66462.384368

Suggestions on improvements

- Use better algoriths like the RandomRegressor
- Use of Neural Nets
- Get more parameters