Housing Estimator

By: Team Test Data

Carol Buckinger, John Gutierrez, Shruti Bansal, and Megan Staley

The Overview

For our project, we decided to explore some machine learning models that would show us how housing characteristics affect the median house value in various neighborhoods in 1990. We gathered four different datasets that contained information such as total rooms & bedrooms as well as average income, location, population, and weather using an open weather map API. We compared a variety of machine learning models including but not limited to Linear Regression, Random Forest Regressor, and Gradient Boosting Regression.

Which model do you think worked best for our project?

How do the features influence home values?

Tools and Technologies

DataBase Integration

- Quick DBD
- AWS RDS
- PgAdmin
- Google Colab
- Pyspark
- Python

Preprocessing

- Jupyter Notebook
- Pandas
- Python
- Numpy
- Citypy
- API

Visualization

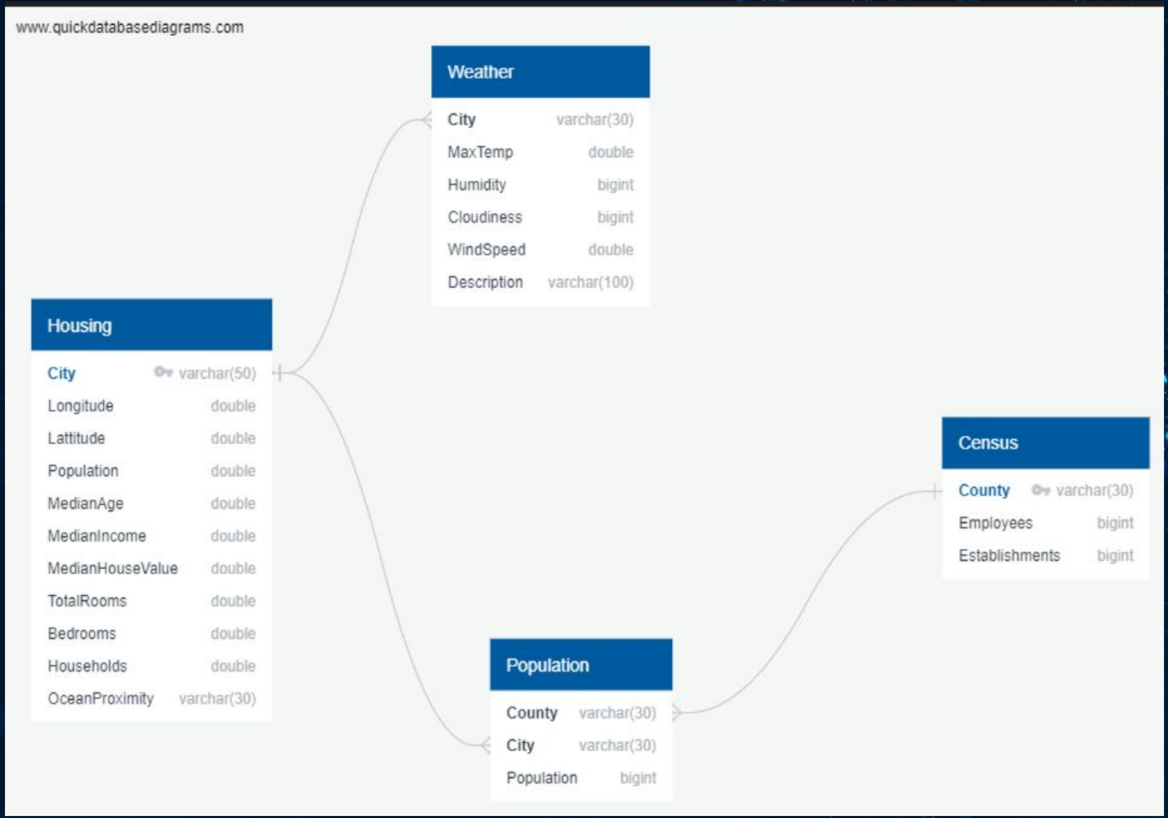
- Tableau
- Matplotlib

ML Model

- Jupyter Notebook
- Python
- Scikit-learn
- Numpy

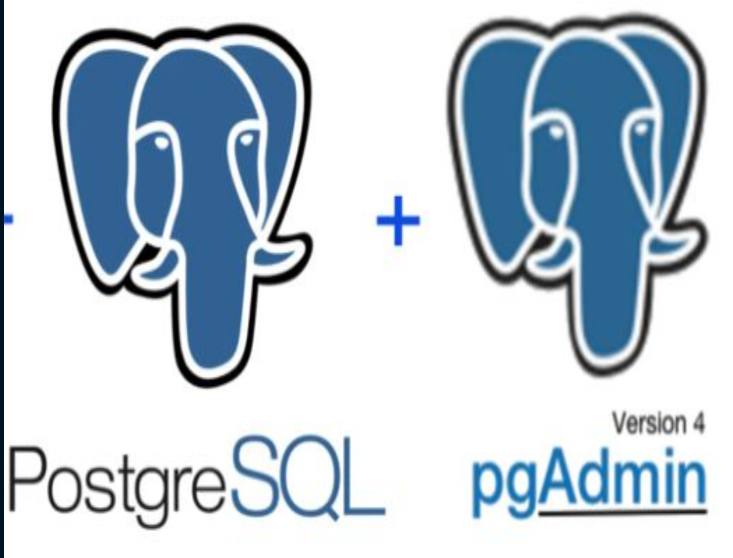
Entity Relationship Diagram

After considering several data sources for the housing data, the team identified the 1990 California Housing Prices database from Kaggle as the main dataset. This dataset is comprehensive, wide-ranging, saturated in geographic area, and includes geographical location coordinates which can link to a wide range of other data sources. The external data for county employment figures were derived from census data (Census.gov) and weather from openweathermap.org, both called using API's. The population information is the Kaggle California cities dataset. After cleaning, restructuring, refining and merging the individual datasets, these four datasets became the production database and subsequently housed in AWS.



Database Integration

PySpark



Dataset

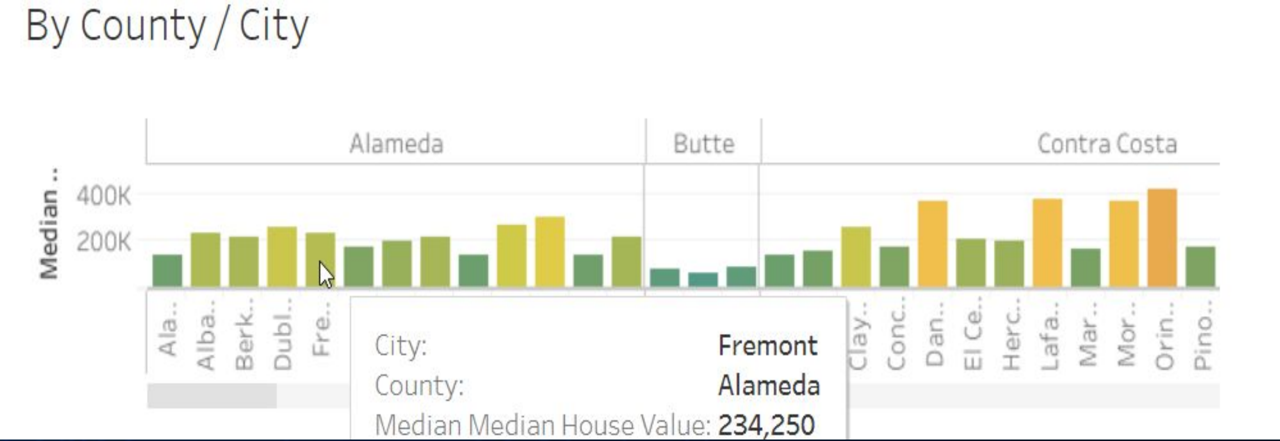
- Longitude
- Latitude
- Median Age
- Median Income
- Median House Value
- Total Rooms
- Bedrooms
- Households
- Ocean Proximity
- City
- County
- Population
- Max Temp
- Humidity
- Cloudiness
- Wind Speed
- Description
- Employees
- Establishments

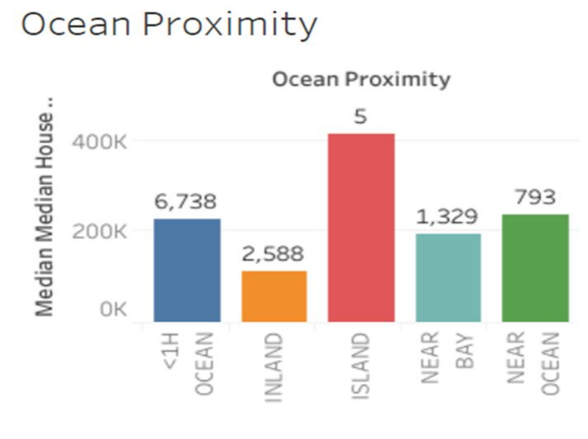
	City	Longitude	latitude	Population	median_age	median_income	median_house_value	total_rooms	Bedrooms	Households
0	Mission Viejo	-117.66	33.61	789	16	8.4112	286900	2022	254	270
1	Mission Viejo	-117.66	33.62	1962	16	6.2177	256600	4065	661	636
2	Mission Viejo	-117.67	33.61	1972	24	5.7871	227400	3859	661	624
3	Mission Viejo	-117.66	33.61	1713	17	6.0471	248400	3464	519	530
4	Mission Viejo	-117.66	33.61	860	21	7.1497	274000	1932	266	286

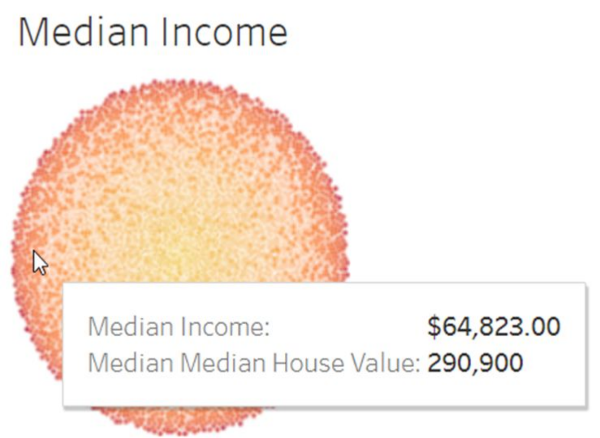
+

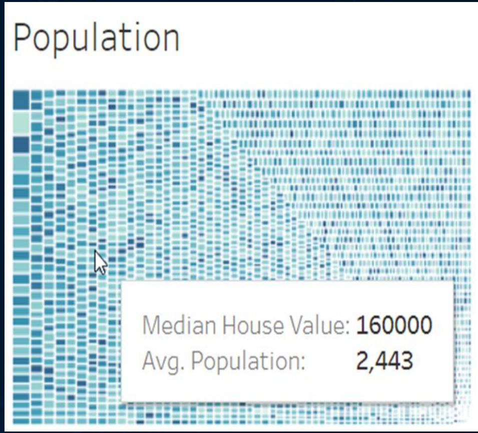
ocean_proximity	max_temp	Humidity	Cloudiness	wind_speed	Description	County	Employees	Establishments
<1H OCEAN	94.75	63	59	5.01	broken clouds	Orange	1191075	71255
<1H OCEAN	94.75	63	59	5.01	broken clouds	Orange	1191075	71255
<1H OCEAN	94.75	63	59	5.01	broken clouds	Orange	1191075	71255
<1H OCEAN	94.75	63	59	5.01	broken clouds	Orange	1191075	71255
<1H OCEAN	94.75	63	59	5.01	broken clouds	Orange	1191075	71255

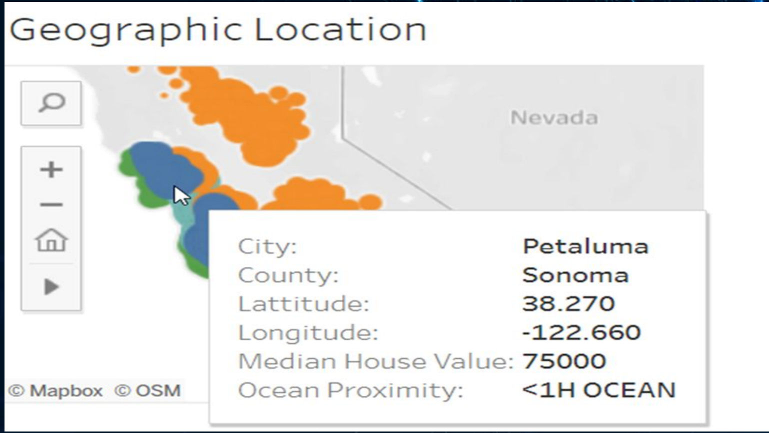
Data Visualization



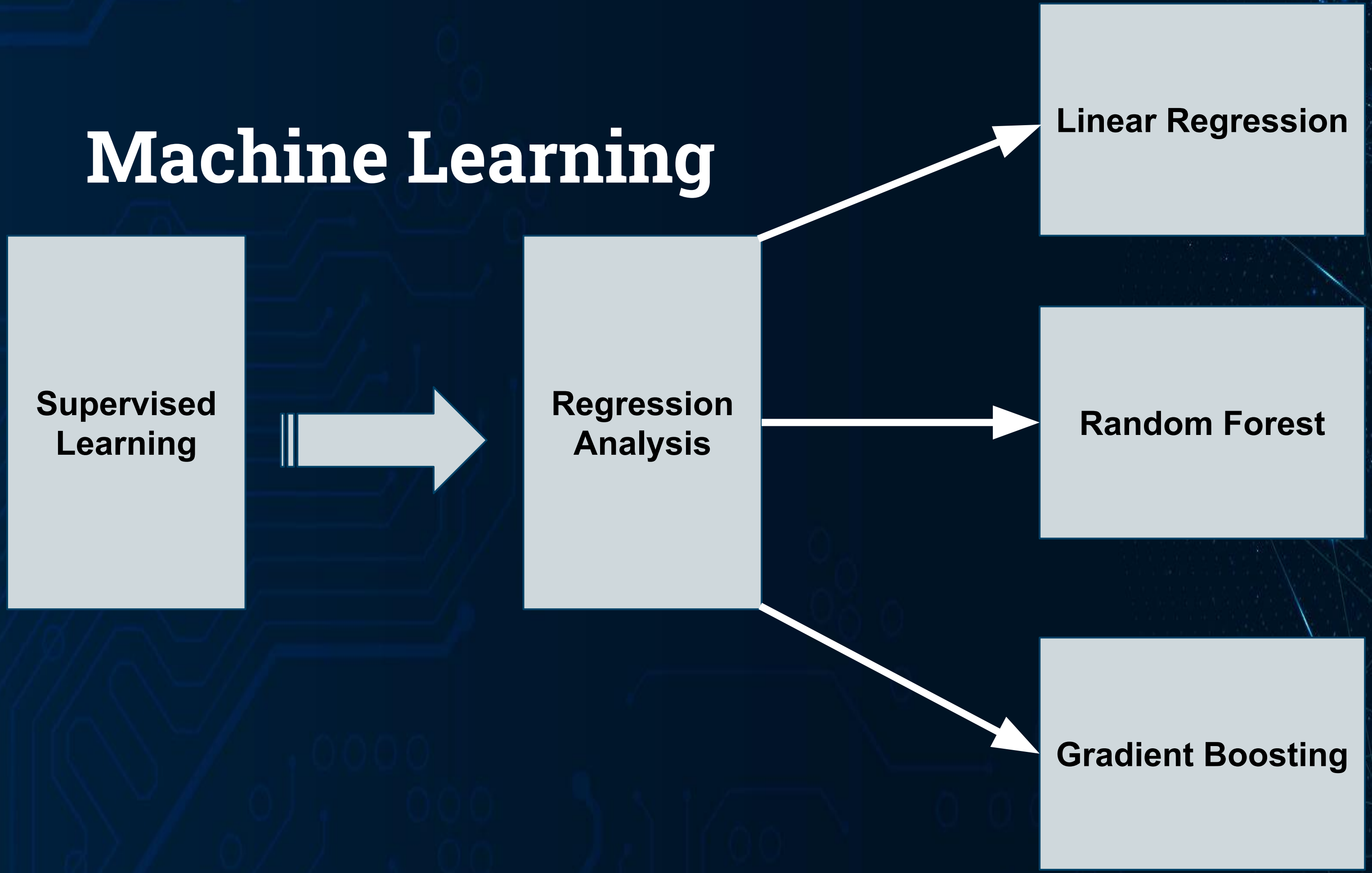








Machine Learning



Base Layout for Machine Learning Modeling

Read Data CSV

Drop Uncorrelated Columns

Encoding

Splitting & Scaling

Base Models

We explored Linear Regression

Only came up with a score of 64.6%

Linear Regression

```
# Define the model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

```
# Fit the model
model.fit(X_train_scaled, y_train)
```

```
LinearRegression()
```

```
print(model.coef_)
print(model.intercept_)
```

```
[-5.69367879e+04  7.85802194e+03  7.04750226e+04  6.11368447e+03
  6.88516828e+03  4.80234909e+04  5.92503369e+03  1.34505638e+04
  1.97150188e+04 -3.15475934e+03  3.16834920e+05 -3.06643388e+05
 -7.52356209e+17 -6.39623928e+17 -2.47361426e+16 -4.84719566e+17
 -3.95203352e+17  3.01199093e+17  5.20520942e+17  4.42966001e+17
  4.95151312e+16  2.08613792e+17  2.50728456e+17  8.12950648e+16
  3.79938159e+16]
225385.53027222757
```

```
X.columns
#'Bedrooms','Households','Employees','ocean_proximity_<1H OCEAN',

coef_df = pd.DataFrame(data= model.coef_,index= X.columns, columns = ['coef_value'])
coef_df
```

	coef_value
Population	-5.693679e+04
median_age	7.858022e+03
median_income	7.047502e+04
total_rooms	6.113684e+03
Bedrooms	6.885168e+03
Households	4.802349e+04
max_temp	5.925034e+03
Humidity	1.345056e+04
Cloudiness	1.971502e+04
wind_speed	-3.154759e+03
Employees	3.168349e+05
Establishments	-3.066434e+05
ocean_proximity_<1H OCEAN	-7.523562e+17
ocean_proximity_INLAND	-6.396239e+17
ocean_proximity_ISLAND	-2.473614e+16
ocean_proximity_NEAR BAY	-4.847196e+17
ocean_proximity_NEAR OCEAN	-3.952034e+17
Description_broken clouds	3.011991e+17
Description_clear sky	5.205209e+17
Description_few clouds	4.429660e+17
Description_haze	4.951513e+16
Description_overcast clouds	2.086138e+17
Description_scattered clouds	2.507285e+17
Description_smoke	8.129506e+16
Description_thunderstorm	3.799382e+16

```
y_pred = model.predict(X_test_scaled)
```

```
results = pd.DataFrame({'Actual':y_test,'Predicted':y_pred})
results
```

	Actual	Predicted
9412	500001	330847.780272
9331	308900	270935.780272
4120	189800	183135.780272
[illegible]	[illegible]	104095.780272
[illegible]	[illegible]	[illegible]
...	...	...
10120	137900	124063.780272
8009	500001	374239.780272
3005	156300	195543.780272
9492	179200	213583.780272
6454	158900	264799.780272

3818 rows × 2 columns

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 49596.080332173035
Mean Squared Error: 4560634611.67735
Root Mean Squared Error: 67532.47079499868
```

```
model.score(X_test_scaled, y_test)
```

```
0.6469528337450026
```

Testing Random Forest Regressor

Random Forest Regressor

```
random_forest = RandomForestRegressor(n_estimators = 1000, random_state = 1, criterion='squared_error',max_features = 'auto',
                                      max_depth = 7)
random_forest.fit(X_train_scaled,y_train)
```

```
RandomForestRegressor(max_depth=7, n_estimators=1000, random_state=1)
```

```
# Calculated actual v. predicted values for y

y_pred = random_forest.predict(X_test_scaled)
print(len(y_pred))
print(len(y_test))

print(f"y_pred ",y_pred)
print(f"y_test ", y_test)


df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
df
```

```
3818
3818
y_pred  [398961.56724012 228462.04722042 182703.48924931 ... 167276.99514246
 226195.14834486 264229.80582515]
y_test  9412     500001
9331     308900
4120     189800
10851    103600
618      366700
          ...
10120    137900
8009     500001
3005     156300
9492     179200
6454     158900
Name: median_house_value, Length: 3818, dtype: int64
```

	Actual	Predicted
9412	500001	398961.567240
9331	308900	228462.047220
4120	189800	182703.489249
10851	103600	83863.092953
618	366700	225737.692548
...	...	...
10120	137900	94539.558996
8009	500001	348411.255977
3005	156300	167276.995142
9492	179200	226195.148345
6454	158900	264229.805825

3818 rows × 2 columns

1 × 1px   2098 × 941px   100%

Score→ 71.2%


```
# Calculated actual v. predicted values for y

y_pred = random_forest.predict(X_test_scaled)
print(len(y_pred))
print(len(y_test))

print(f"y_pred ",y_pred)
print(f"y_test ", y_test)


df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
df
```

```
3818
3818
y_pred  [398961.56724012 228462.04722042 182703.48924931 ... 167276.99514246
 226195.14834486 264229.80582515]
y_test  9412     500001
9331     308900
4120     189800
10851    103600
618      366700
          ...
10120    137900
8009     500001
3005     156300
9492     179200
6454     158900
Name: median_house_value, Length: 3818, dtype: int64
```

	Actual	Predicted
9412	500001	398961.567240
9331	308900	228462.047220
4120	189800	182703.489249
10851	103600	83863.092953
618	366700	225737.692548
...	...	...
10120	137900	94539.558996
8009	500001	348411.255977
3005	156300	167276.995142
9492	179200	226195.148345
6454	158900	264229.805825

3818 rows × 2 columns


```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 43042.8736477768
Mean Squared Error: 3723029129.806957
Root Mean Squared Error: 61016.62994468767
```

```
random_forest.score(X_test_scaled, y_test)
```

```
0.7117934243629898
```

Testing Gradient Boosting Regressor

```python
from sklearn.ensemble import GradientBoostingRegressor
```

```python
# define the model
model = GradientBoostingRegressor(random_state = 1)
model.fit(X_train_scaled,y_train)
```

```
GradientBoostingRegressor(random_state=1)
```

```python
y_pred = model.predict(X_test_scaled)
y_pred

df1=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
df
```

	Actual	Predicted
9412	500001	398961.567240
9331	308900	228462.047220
4120	189800	182703.489249
10851	103600	83863.092953
618	366700	225737.692548
...	...	...
10120	137900	94539.558996
8009	500001	348411.255977
3005	156300	167276.995142
9492	179200	226195.148345
6454	158900	264229.805825

Score→ 74.9%

	Actual	Predicted
9412	500001	398961.567240
9331	308900	228462.047220
4120	189800	182703.489249
10851	103600	83863.092953
618	366700	225737.692548
...	...	...
10120	137900	94539.558996
8009	500001	348411.255977
3005	156300	167276.995142
9492	179200	226195.148345
6454	158900	264229.805825

3818 rows × 2 columns

```python
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 40199.80440804308
Mean Squared Error: 3229866156.186027
Root Mean Squared Error: 56831.911424709506
```

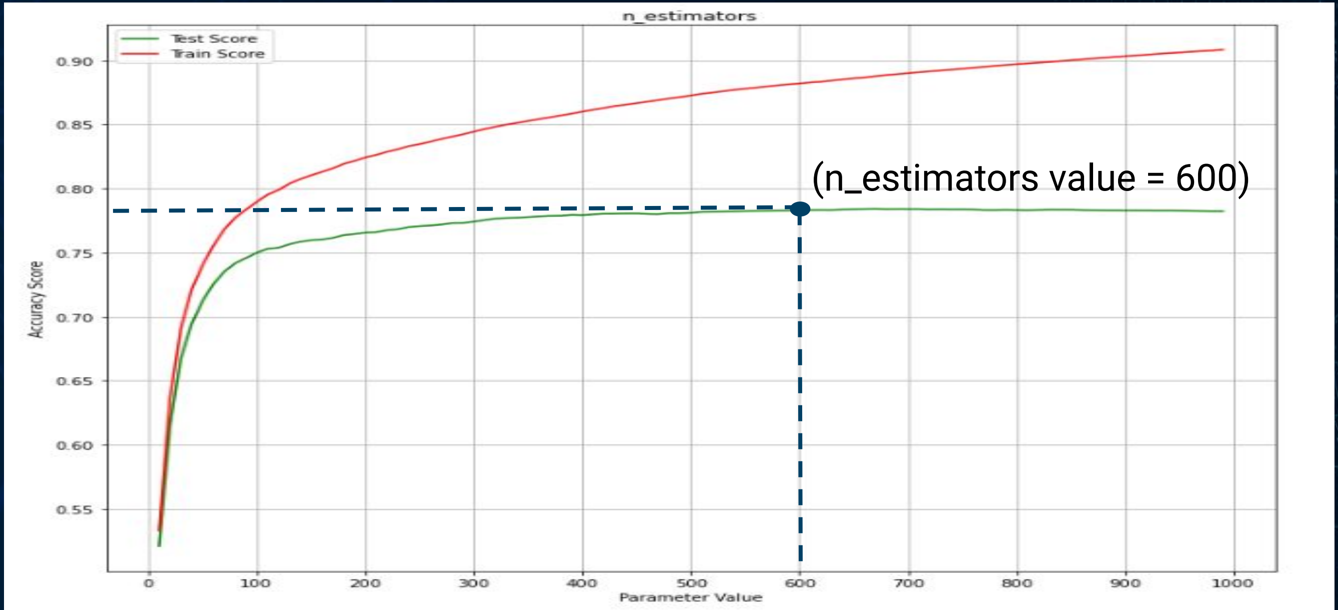
```python
model.score(X_test_scaled, y_test)
```

```
0.7499700829124283
```

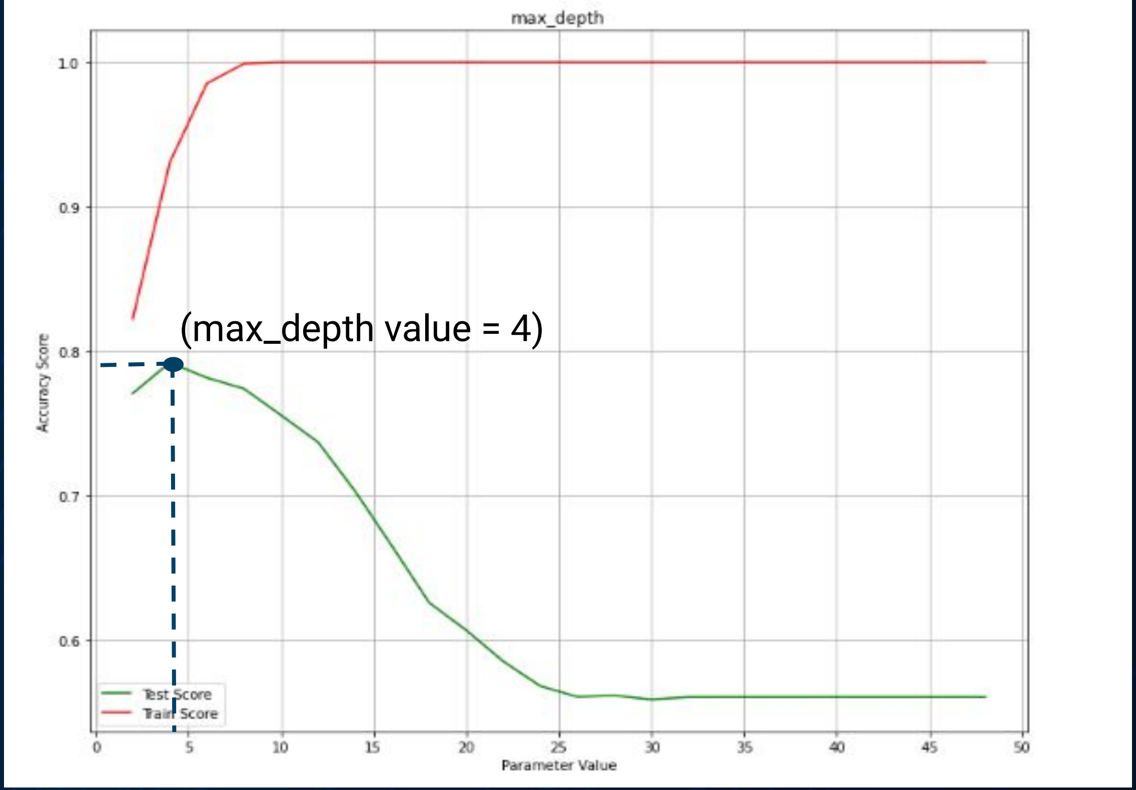
Gradient Boosting Regression

n_estimators

n_estimators indicate the total number of trees used in the model to arrive at the final result. Higher number of trees gives us better performance but makes our code slower. As seen in the graph below, the accuracy score over the training set increases continuously with the increase in the hyper parameter value. On the other hand as the n_estimators value increases the performance over the test set increases initially but after the n_estimators value 600 accuracy score becomes stagnant. Which means even if we increase the value of n_estimators above 600 there is no major improvement in the accuracy level of the test set. Since we were not gaining much improvement in the model's performance after a certain point, increase in the n_estimators value will not add any value to the model and will also make our model slower and probably overfitted too.



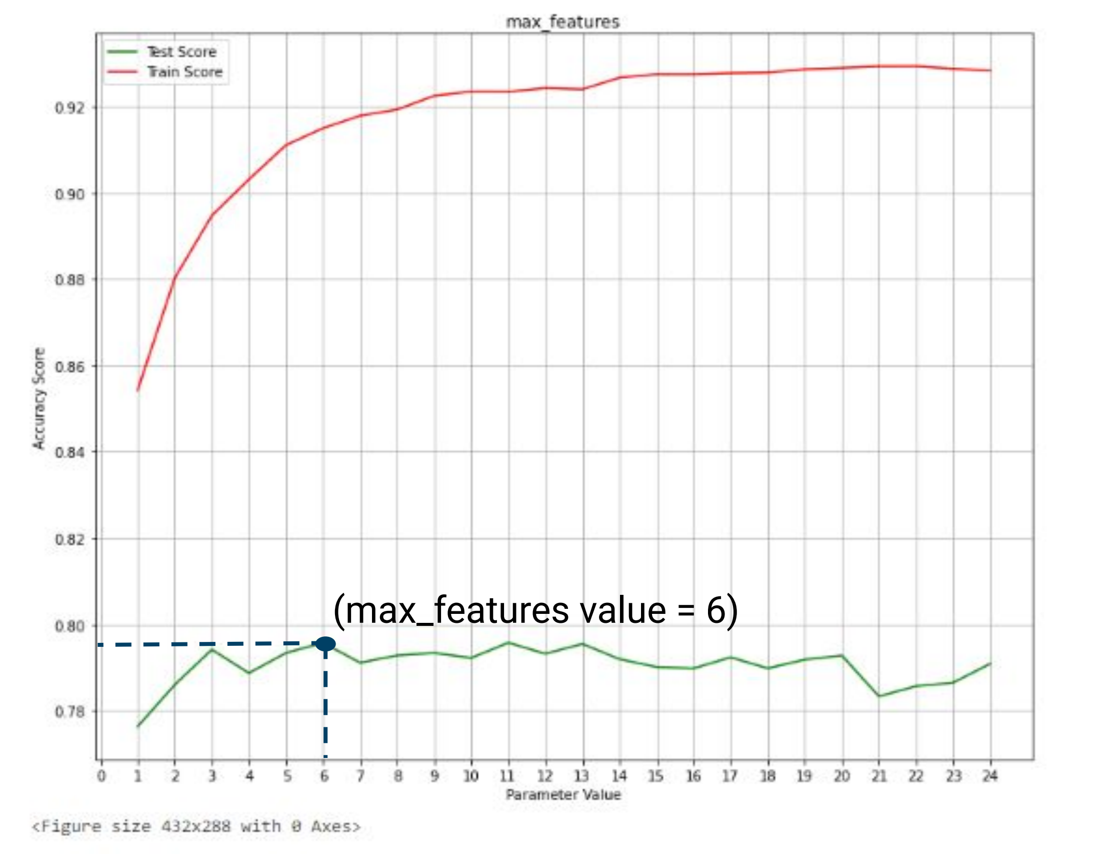
max_depth



max_depth indicates the maximum depth of a tree in the model. It is basically defined as longest path between the root node and the leaf node. Using the max depth, we can limit up to what depth we want every tree to grow. In the graph, we can see that as the value of max depth increases, the performance of the model over training set increases continuously and eventually achieves the 100% accuracy score. On the other hand as the max_depth value increases, the performance over the test set increases initially but after max-depth value 4, it starts to decrease rapidly. Which means that the tree starts to overfit the training set and therefore is not able to generalize over the unseen points in the test set.

max_features

max_features simulates the number of maximum features provided to each tree in a model. The model chooses some random samples from the features to find the best split. In the graph, we can see that as the value of max_features increases, the performance of the model over training set increases continuously. On the other hand as the max_features value increases , the performance over the test set increases initially but after max_features value 6, performance is not showing much improvement. Which means that the tree starts to overfit the training set and hence is not able to generalize over the unseen points in the test set.



Gradient Boosting Regressor

```python
# Split our preprocessed data into our features and target arrays
X = housing_df.drop(columns = ["median_house_value"])
y = housing_df['median_house_value']
```

```python
# Split the preprocessed data into a training and testing dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1, test_size=1/3)
```

```python
# Creating a StandardScaler instance.
scaler = StandardScaler()
# Fitting the Standard Scaler with the training data.
X_scaler = scaler.fit(X_train)

# Scaling the data.
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

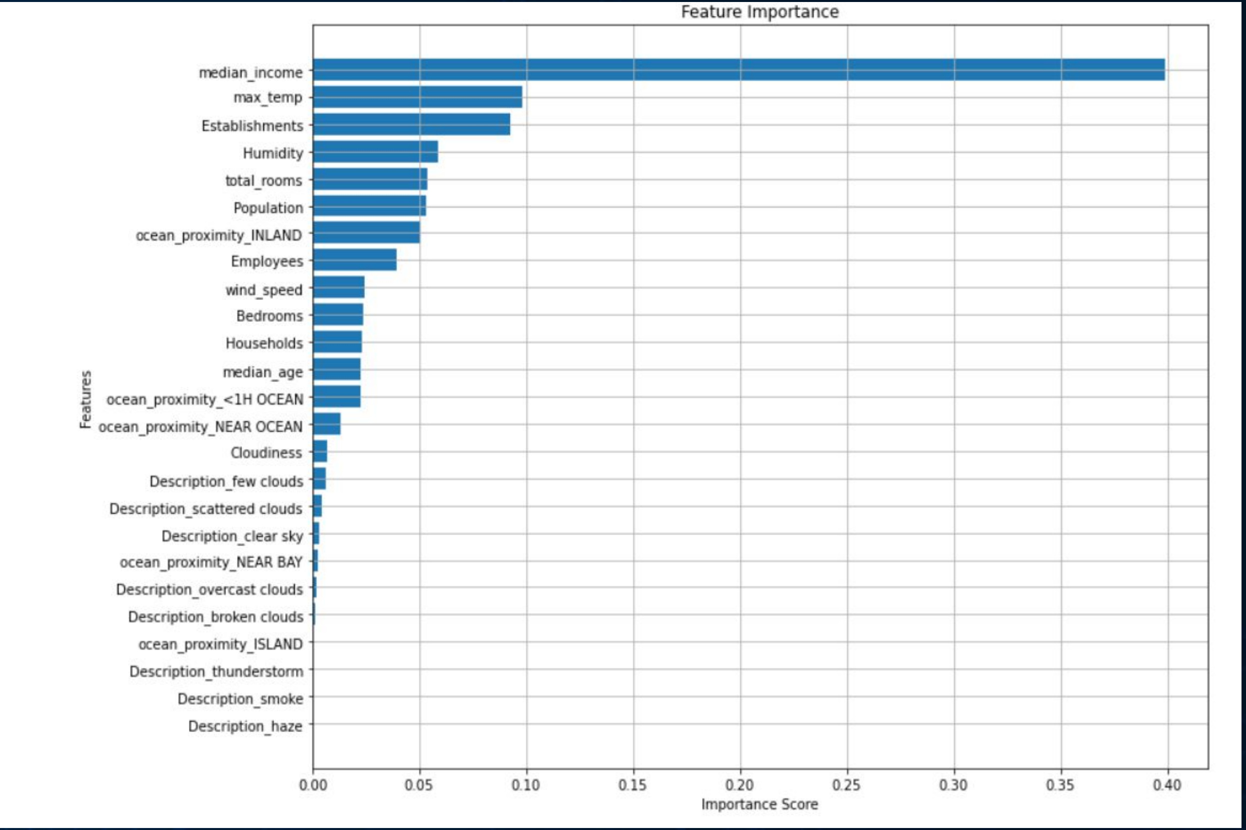
```python
params = {
        "n_estimators": 600,
    "max_depth": 4,
    "max_features": 6,
    "random_state": 1
     }
model = GradientBoostingRegressor(**params)
model.fit(X_train_scaled,y_train)

y_pred = model.predict(X_test_scaled)
print(f'model score is :{model.score(X_test_scaled, y_test)}')
```

```
model score is :0.7957493466432817
```

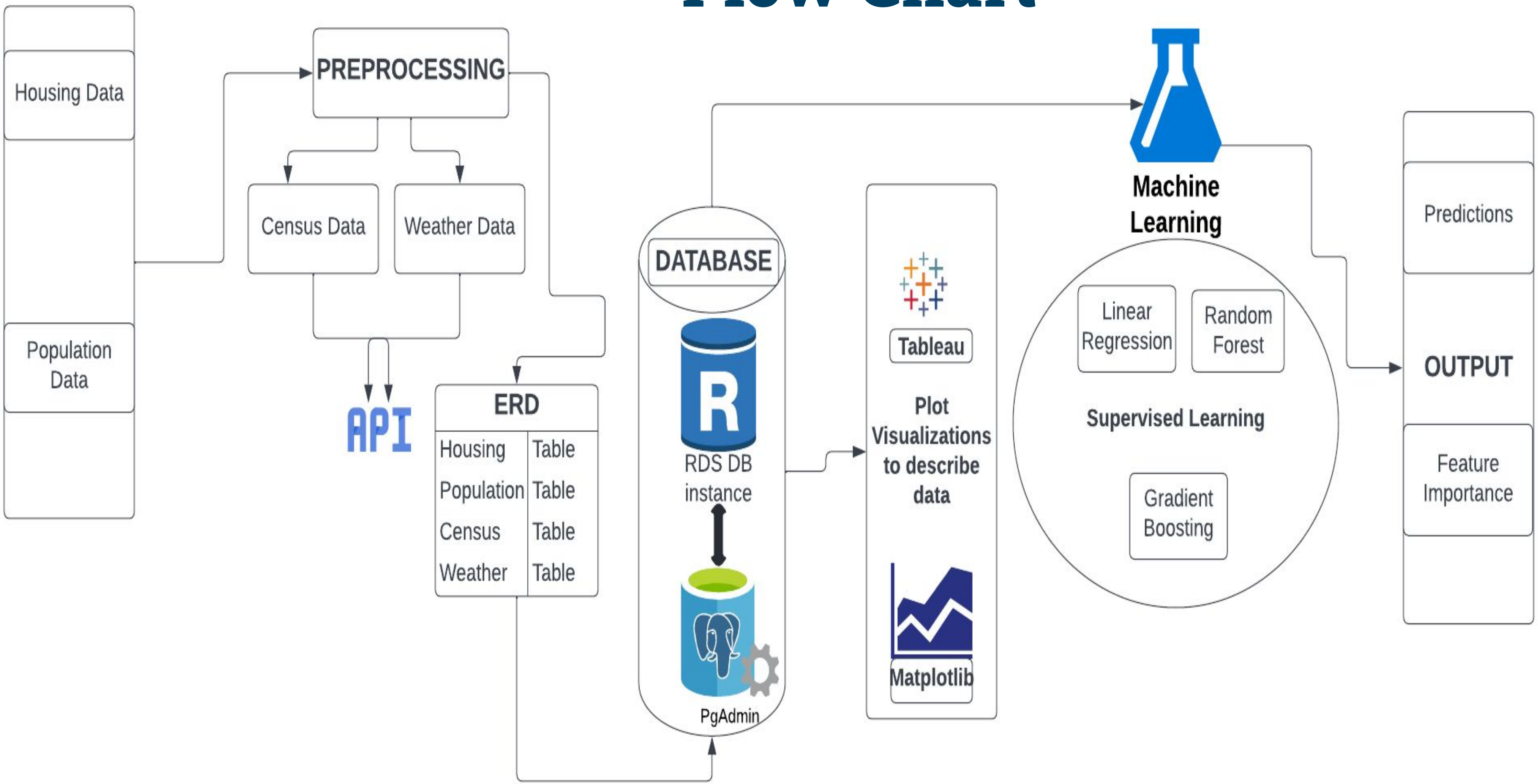
Optimized Model Score 79.6%

Feature Importance



We have concluded that using Gradient Boosting regression is the most appropriate model. Overall median income came out as the top influencer with a weighted score of 40%. Followed by max temperature and the number of business establishments in the county had a score around 10%. Humidity, total rooms, population, and inland/ocean proximity also had a small impact on the housing prices. The number of employed people, wind speed, total bedrooms, amount of households, age and near ocean had minimal impact. All other features had a little to no weighted score on housing prices.

Flow Chart



Housing Data
Population Data
PREPROCESSING
Census Data
Weather Data
API
ERD
Housing Table
Population Table
Census Table
Weather Table
DATABASE
RDS DB instance
PgAdmin
Tableau
Plot Visualizations to describe data
Matplotlib
Machine Learning
Linear Regression
Random Forest
Supervised Learning
Gradient Boosting
Predictions
OUTPUT
Feature Importance

Recommendations

- We would recommend more preliminary data exploration. Also, more features engineering based on the data exploration.

- Add more context variables. Including community crime data.

Sources

- Housing Data: https://www.kaggle.com/datasets/camnugent/california-housing-prices
- Population Data: https://www.kaggle.com/datasets/camnugent/california-housing-feature-engineering?select=cal_populations_city.csv
- Census Data: https://api.census.gov/data/1990/cbp?get=GEO_TTL,EMP,ESTAB&for=county:*&in=state:06&key=
- Weather Data: http://openweathermap.org/

GitHub and Dashboard links

- GitHub:

  https://github.com/jsguti323/Housing_Estimator

- Tableau:
  https://public.tableau.com/app/profile/john.gutierrez7405/viz/Housing_Estimator/Housing_Estimator


ANY
QUESTIONS?