In this Python notebook, I will be analyzing the dataset to see if there are any features or patterns I can leverage to better build my model, which aims to predict median house values. I will be using `numpy`, `pandas`, and `seaborn` to help me go through the data. For model implementation, I will be using XGBoost. The dataset used is the California housing data set, which can be found on Kaggle here: https://www.kaggle.com/datasets/camnugent/california-housing-prices

```
# Necessary imports
import numpy as np
import pandas as pd
import seaborn as sns
```

As this is a CSV file, I will first load the file into a `pandas` dataframe and will take a look at the format before deciding what to do next.

```
# Read CSV file into dataframe
df = pd.read_csv("housing.csv")

df
```

```
       longitude  latitude  housing_median_age  total_rooms
total_bedrooms  \
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

       population  households  median_income  median_house_value  \
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
```

```
3            558.0       219.0          5.6431           341300.0
4            565.0       259.0          3.8462           342200.0
...            ...         ...             ...                ...
20635        845.0       330.0          1.5603            78100.0
20636        356.0       114.0          2.5568            77100.0
20637       1007.0       433.0          1.7000            92300.0
20638        741.0       349.0          1.8672            84700.0
20639       1387.0       530.0          2.3886            89400.0

      ocean_proximity
0            NEAR BAY
1            NEAR BAY
2            NEAR BAY
3            NEAR BAY
4            NEAR BAY
...               ...
20635          INLAND
20636          INLAND
20637          INLAND
20638          INLAND
20639          INLAND

[20640 rows x 10 columns]
```

The next step is to clean the data. There is potential for missing data, so the dataframe must be checked for any rows at are missing any data. If there are very few in comparison to the total number of rows (20640), then the rows will be entirely removed.

```
# Figure out how many rows of missing data there is
missing_rows = df[df.isnull().any(axis=1)]

missing_rows

       longitude  latitude  housing_median_age  total_rooms
total_bedrooms  \
290      -122.16     37.77                47.0       1256.0
NaN
341      -122.17     37.75                38.0        992.0
NaN
538      -122.28     37.78                29.0       5154.0
NaN
563      -122.24     37.75                45.0        891.0
NaN
696      -122.10     37.69                41.0        746.0
NaN
...          ...       ...                 ...          ...
...
20267    -119.19     34.20                18.0       3620.0
NaN
```

```
20268    -119.18     34.19                19.0       2393.0
NaN
20372    -118.88     34.17                15.0       4260.0
NaN
20460    -118.75     34.29                17.0       5512.0
NaN
20484    -118.72     34.28                17.0       3051.0
NaN

       population  households  median_income  median_house_value  \
290         570.0       218.0         4.3750            161900.0
341         732.0       259.0         1.6196             85100.0
538        3741.0      1273.0         2.5762            173400.0
563         384.0       146.0         4.9489            247100.0
696         387.0       161.0         3.9063            178400.0
...           ...         ...            ...                 ...
20267      3171.0       779.0         3.3409            220500.0
20268      1938.0       762.0         1.6953            167400.0
20372      1701.0       669.0         5.1033            410700.0
20460      2734.0       814.0         6.6073            258100.0
20484      1705.0       495.0         5.7376            218600.0

      ocean_proximity
290          NEAR BAY
341          NEAR BAY
538          NEAR BAY
563          NEAR BAY
696          NEAR BAY
...               ...
20267      NEAR OCEAN
20268      NEAR OCEAN
20372       <1H OCEAN
20460       <1H OCEAN
20484       <1H OCEAN

[207 rows x 10 columns]
```

There are only 207 rows with missing data, so it should be okay to remove them.

```
df = df.dropna()

df

       longitude  latitude  housing_median_age  total_rooms
total_bedrooms  \
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

       population  households  median_income  median_house_value  \
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
3           558.0       219.0         5.6431            341300.0
4           565.0       259.0         3.8462            342200.0
...           ...         ...            ...                 ...
20635       845.0       330.0         1.5603             78100.0
20636       356.0       114.0         2.5568             77100.0
20637      1007.0       433.0         1.7000             92300.0
20638       741.0       349.0         1.8672             84700.0
20639      1387.0       530.0         2.3886             89400.0

      ocean_proximity
0            NEAR BAY
1            NEAR BAY
2            NEAR BAY
3            NEAR BAY
4            NEAR BAY
...               ...
20635          INLAND
20636          INLAND
20637          INLAND
20638          INLAND
20639          INLAND

[20433 rows x 10 columns]
```

This will one-hot encode the ocean proximity column so that it can be better analyzed by the model (generally numeric values are preferred).

```python
df = pd.get_dummies(df, columns=["ocean_proximity"], drop_first=True)

df
```

```
       longitude  latitude  housing_median_age  total_rooms
total_bedrooms  \
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

       population  households  median_income  median_house_value  \
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
3           558.0       219.0         5.6431            341300.0
4           565.0       259.0         3.8462            342200.0
...           ...         ...            ...                 ...
20635       845.0       330.0         1.5603             78100.0
20636       356.0       114.0         2.5568             77100.0
20637      1007.0       433.0         1.7000             92300.0
20638       741.0       349.0         1.8672             84700.0
20639      1387.0       530.0         2.3886             89400.0

       ocean_proximity_INLAND  ocean_proximity_ISLAND  \
0                       False                   False
1                       False                   False
2                       False                   False
3                       False                   False
4                       False                   False
...                       ...                     ...
20635                    True                   False
```

```
20636                    True                       False
20637                    True                       False
20638                    True                       False
20639                    True                       False

       ocean_proximity_NEAR BAY  ocean_proximity_NEAR OCEAN
0                          True                       False
1                          True                       False
2                          True                       False
3                          True                       False
4                          True                       False
...                         ...                         ...
20635                     False                       False
20636                     False                       False
20637                     False                       False
20638                     False                       False
20639                     False                       False

[20433 rows x 13 columns]
```

This is the inital model that will be used as a baseline:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor


def run_model(df):
    x = df.drop(columns=["median_house_value"])
    y = df["median_house_value"]

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=419)

    # Initialize the XGBoost regressor
    original_model = XGBRegressor(objective="reg:squarederror",
n_estimators=1000, learning_rate=0.01, max_depth=7)

    # Train the model
    original_model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = original_model.predict(X_test)

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
```

```
    print(f"Mean Squared Error: ${mse:.2f}")
    print(f"Mean Absolute Error: ${mae:.2f}")
    print(f"R^2 Score: {r2}")

    # Medical tolerance (15%)
    tolerance = 0.15

    # Calculate accuracy
    accuracy = ((abs(y_pred - y_test) / y_test) <= tolerance).mean()
    print(f"Accuracy within {tolerance * 100}% tolerance:
{accuracy:.2%}")

    return {"model": original_model, "mae": mae, "mse": mse, "r^2":
r2, "accuracy": accuracy}

original_data = run_model(df)
```

```
Mean Squared Error: $2267413855.57
Mean Absolute Error: $31261.91
R^2 Score: 0.832266042287044
Accuracy within 15.0% tolerance: 59.70%
```

Data visualization is necessary for identifying what parameters are desireable for the model and if there are any data distributions to take care of (whether the data is very dense or spread apart and whether there are any significant outliers that can skew the data).

Since median house value is one of the columns in the data, comparing the other parameters against the median house value will be good for checking for correlation. First, we will check the importance of the different features.

```
columns = df.drop(columns=["median_house_value"]).columns
column_importance = dict(zip(columns, [float(val) for val in
original_data["model"].feature_importances_]))

sorted_col_importance = dict(sorted(column_importance.items(),
key=lambda item: item[1], reverse=True))

sorted_col_importance
```

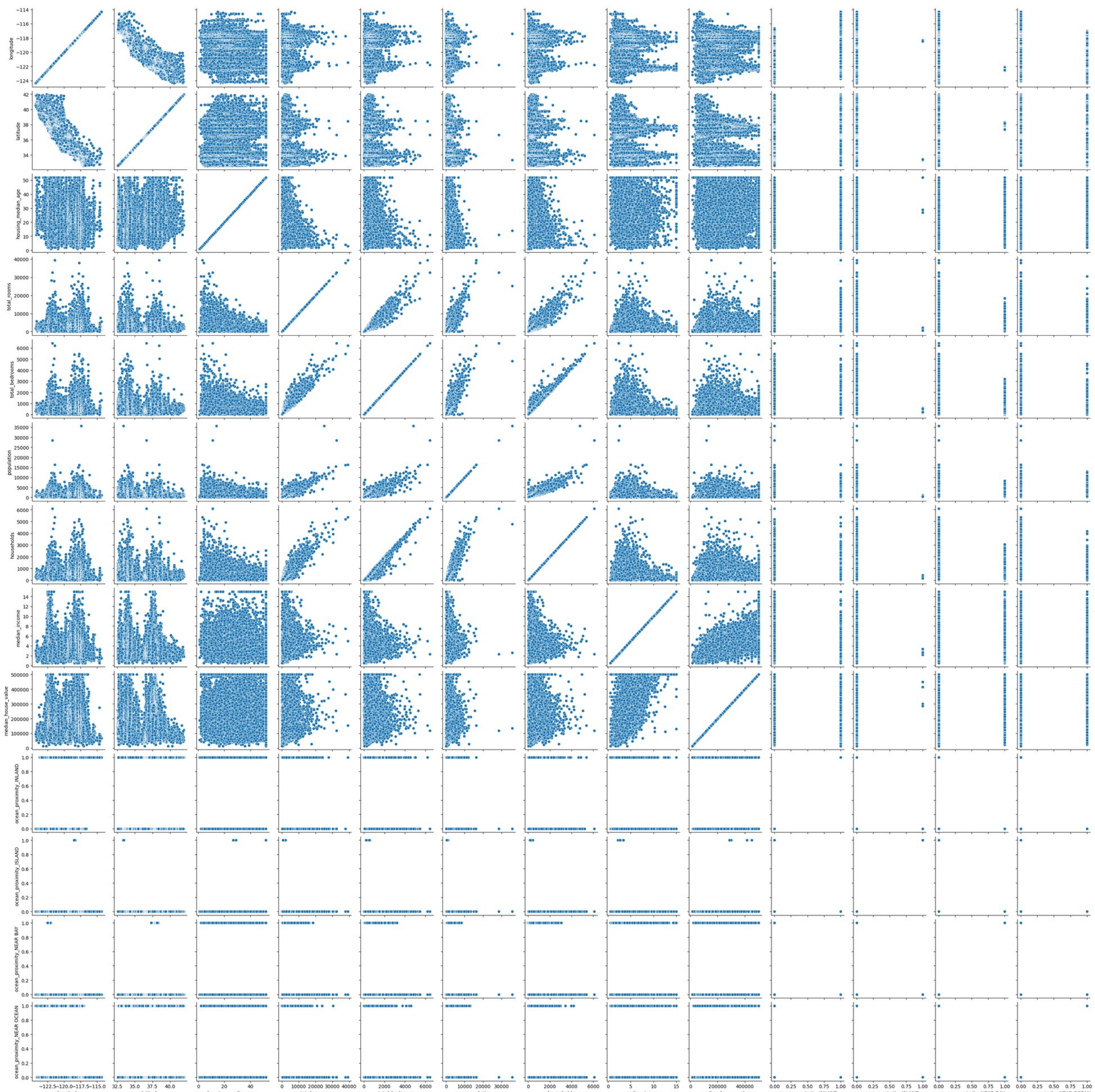
```
{'ocean_proximity_INLAND': 0.5352905988693237,
 'median_income': 0.22805730998516083,
 'ocean_proximity_NEAR OCEAN': 0.038734279572963715,
 'ocean_proximity_ISLAND': 0.03855923190712929,
 'latitude': 0.029591072350740433,
 'longitude': 0.027198996394872665,
 'housing_median_age': 0.024348242208361626,
 'population': 0.019737565889954567,
 'total_bedrooms': 0.019181597977876663,
 'ocean_proximity_NEAR BAY': 0.016545208171010017,
```

```
 'households': 0.013357114978134632,
 'total_rooms': 0.009398694150149822}
```

We can see that `ocean_proxmity_INLAND` and `median_income` are the two top features while `households` and `total_rooms` are not particularly helpful. Before removing or adding features, let's compare the categories with one another with a pairplot and a correlation heat map.

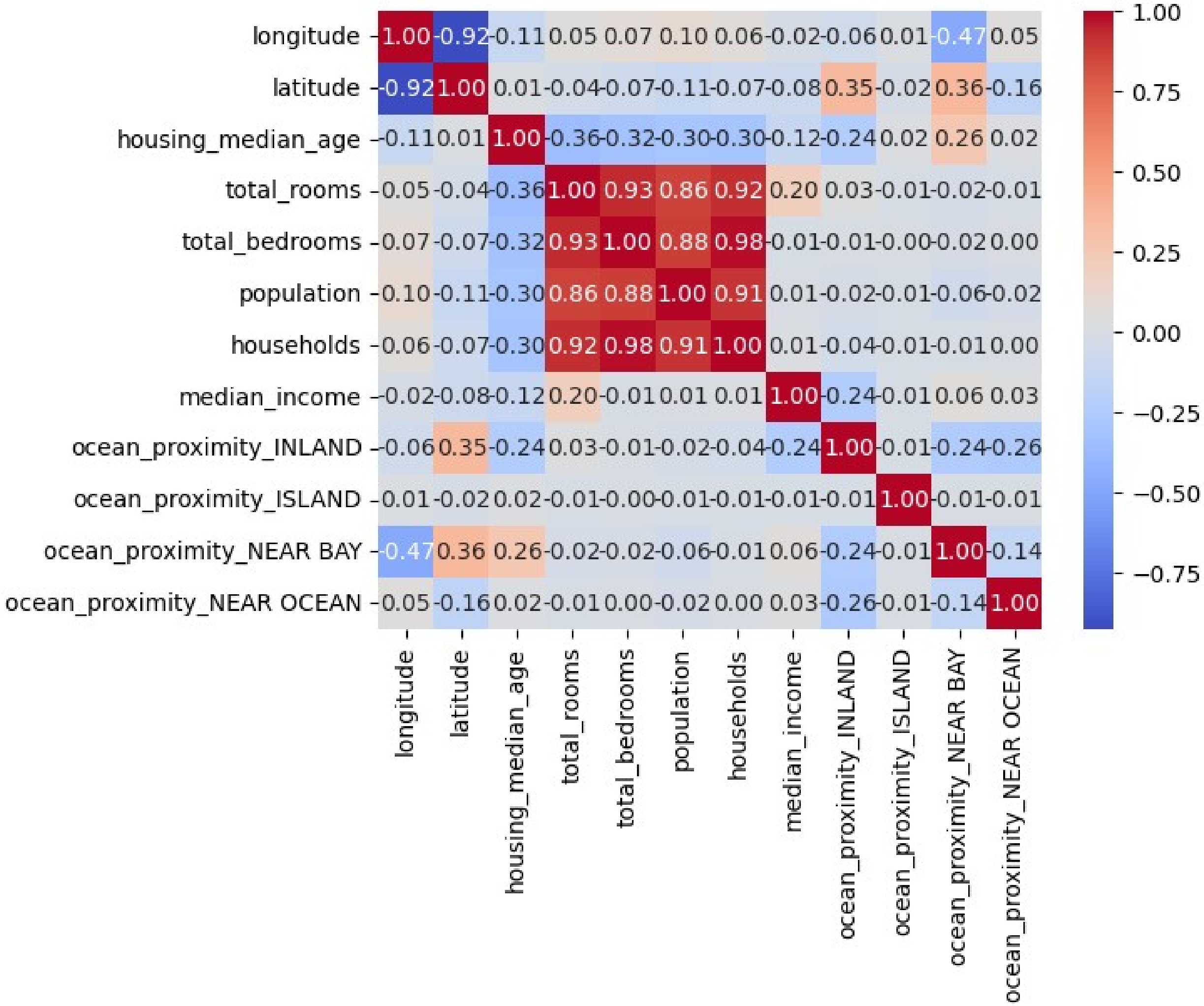
```
pairplot = sns.pairplot(df, diag_kind=None)

pairplot
```

```
<seaborn.axisgrid.PairGrid at 0x1bcfe42d940>
```




```
sns.heatmap(df.drop(columns="median_house_value").corr(), annot=True,
cmap="coolwarm", fmt=".2f")
```

```
<Axes: >
```



`total_rooms`, `total_bedrooms`, `population`, and `households` are all very correlated with one another (> 0.90), and are all in the bottom five important features:

- `population`: 0.019737565889954567
- `total_bedrooms`: 0.019181597977876663
- `households`: 0.013357114978134632
- `total_rooms`: 0.009398694150149822

It may be wise to make new features that represent other attributes that may influence house value. Population density (`population` / `households`) could be useful along with rooms per house (average house size, `total_rooms` / `households`)

```
df_v2 = df.copy()

df_v2["pop_density"] = df_v2["population"] / df_v2["households"]
df_v2["house_density"] = df_v2["total_rooms"] / df_v2["households"]

df_v2
```

```
       longitude  latitude  housing_median_age  total_rooms  
total_bedrooms  \
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   

       population  households  median_income  median_house_value  \
0           322.0       126.0         8.3252            452600.0   
1          2401.0      1138.0         8.3014            358500.0   
2           496.0       177.0         7.2574            352100.0   
3           558.0       219.0         5.6431            341300.0   
4           565.0       259.0         3.8462            342200.0   
...           ...         ...            ...                 ...   
20635       845.0       330.0         1.5603             78100.0   
20636       356.0       114.0         2.5568             77100.0   
20637      1007.0       433.0         1.7000             92300.0   
20638       741.0       349.0         1.8672             84700.0   
20639      1387.0       530.0         2.3886             89400.0   

       ocean_proximity_INLAND  ocean_proximity_ISLAND  \
0                       False                   False   
1                       False                   False   
2                       False                   False   
3                       False                   False   
4                       False                   False   
...                       ...                     ...   
20635                    True                   False   
20636                    True                   False   
20637                    True                   False   
20638                    True                   False   
20639                    True                   False   
```

```
       ocean_proximity_NEAR BAY  ocean_proximity_NEAR OCEAN
pop_density  \
0                          True                       False
2.555556
1                          True                       False
2.109842
2                          True                       False
2.802260
3                          True                       False
2.547945
4                          True                       False
2.181467
...                         ...                         ...          .
..
20635                     False                       False
2.560606
20636                     False                       False
3.122807
20637                     False                       False
2.325635
20638                     False                       False
2.123209
20639                     False                       False
2.616981

       house_density
0           6.984127
1           6.238137
2           8.288136
3           5.817352
4           6.281853
...              ...
20635       5.045455
20636       6.114035
20637       5.205543
20638       5.329513
20639       5.254717

[20433 rows x 15 columns]
```

Now we try the model on the new data.

```
v2_data = run_model(df_v2)

Mean Squared Error: $2189646288.23
Mean Absolute Error: $30181.74
R^2 Score: 0.838018967285961
Accuracy within 15.0% tolerance: 61.46%
```

From here, we can try to see if there is improvement on dropping any features whose importance is too few. We will check the feature importance again and decide on dropping features for the v3 model.

```
columns = df_v2.drop(columns=["median_house_value"]).columns
column_importance = dict(zip(columns, [float(val) for val in
v2_data["model"].feature_importances_]))

sorted_col_importance = dict(sorted(column_importance.items(),
key=lambda item: item[1], reverse=True))

sorted_col_importance
```

```
{'ocean_proximity_INLAND': 0.5162120461463928,
 'median_income': 0.25450021028518677,
 'pop_density': 0.05797304958105087,
 'ocean_proximity_ISLAND': 0.024760089814662933,
 'housing_median_age': 0.024340640753507614,
 'latitude': 0.02215234749019146,
 'longitude': 0.020298369228839874,
 'ocean_proximity_NEAR OCEAN': 0.018944622948765755,
 'ocean_proximity_NEAR BAY': 0.013564365915954113,
 'house_density': 0.012222538702189922,
 'total_bedrooms': 0.010043910704553127,
 'households': 0.009095135144889355,
 'population': 0.008325847797095776,
 'total_rooms': 0.007566750515252352}
```

As `pop_density` and `house_density` both out-weigh the four highly correlated, low importance features, for v3, we can drop `total_bedrooms`, `households`, `population`, and `total_rooms`.

```
df_v3 = df_v2.copy()

# Drop unimportant features
unimportant_features = ["total_bedrooms", "households", "population",
"total_rooms"]
df_v3 = df_v3.drop(columns=unimportant_features)

df_v3
```

```
       longitude  latitude  housing_median_age  median_income  \
0        -122.23     37.88                41.0         8.3252
1        -122.22     37.86                21.0         8.3014
2        -122.24     37.85                52.0         7.2574
3        -122.25     37.85                52.0         5.6431
4        -122.25     37.85                52.0         3.8462
...          ...       ...                 ...            ...
20635    -121.09     39.48                25.0         1.5603
20636    -121.21     39.49                18.0         2.5568
```

```
20637    -121.22     39.43                17.0         1.7000
20638    -121.32     39.43                18.0         1.8672
20639    -121.24     39.37                16.0         2.3886

       median_house_value  ocean_proximity_INLAND
ocean_proximity_ISLAND  \
0                452600.0                   False
False
1                358500.0                   False
False
2                352100.0                   False
False
3                341300.0                   False
False
4                342200.0                   False
False
...                   ...                     ...
...
20635             78100.0                    True
False
20636             77100.0                    True
False
20637             92300.0                    True
False
20638             84700.0                    True
False
20639             89400.0                    True
False

       ocean_proximity_NEAR BAY  ocean_proximity_NEAR OCEAN
pop_density  \
0                          True                       False
2.555556
1                          True                       False
2.109842
2                          True                       False
2.802260
3                          True                       False
2.547945
4                          True                       False
2.181467
...                         ...                         ...          .
..
20635                     False                       False
2.560606
20636                     False                       False
3.122807
20637                     False                       False
2.325635
```

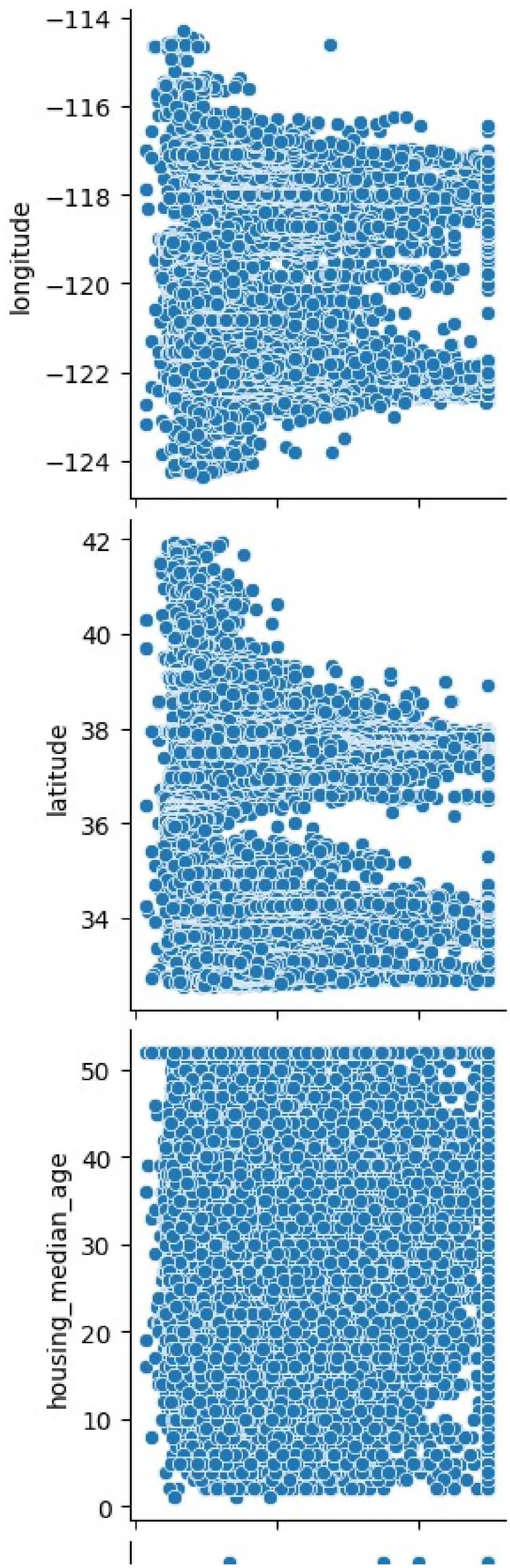
```
20638                          False                              False
2.123209
20639                          False                              False
2.616981

       house_density
0           6.984127
1           6.238137
2           8.288136
3           5.817352
4           6.281853
...              ...
20635       5.045455
20636       6.114035
20637       5.205543
20638       5.329513
20639       5.254717

[20433 rows x 11 columns]
```

Now let's try running the model to see what happens.

```
v3_data = run_model(df_v3)

Mean Squared Error: $2135919585.43
Mean Absolute Error: $29871.71
R^2 Score: 0.84199344793648
Accuracy within 15.0% tolerance: 61.83%
```

There was slight improvement but not much at all. Let's visualize the data now and see if there are any outliers to take care of.

```
features_to_compare =
df_v3.drop(columns=["median_house_value"]).columns

# Create the pairplot
sns.pairplot(df_v3, y_vars=features_to_compare,
x_vars="median_house_value", height=3, aspect=1, kind="scatter")

<seaborn.axisgrid.PairGrid at 0x1bd02880170>
```


longitude
−114
−116
−118
−120
−122
−124
latitude
42
40
38
36
34
housing_median_age
50
40
30
20
10
0

From this, we can see that there are clear outliers:

- `ocean_proximity_ISLAND`: 4 outliers (only ones at 1.0). This is a one-hot encoded value though, so it is better to drop this feature altogether
- `pop_density`: 4 outliers (> 200). These points will just be removed.
- `house_density`: 10 outliers (> 40). These points will just be removed.

```
cleaned_v3 = df_v3.copy()

# Remove outliers
cleaned_v3 = cleaned_v3.drop(columns="ocean_proximity_ISLAND")
cleaned_v3 = cleaned_v3[cleaned_v3["pop_density"] <= 200]
cleaned_v3 = cleaned_v3[cleaned_v3["house_density"] <= 40]

cleaned_v3
```

```
       longitude  latitude  housing_median_age  median_income  \
0        -122.23     37.88                41.0         8.3252
1        -122.22     37.86                21.0         8.3014
2        -122.24     37.85                52.0         7.2574
3        -122.25     37.85                52.0         5.6431
4        -122.25     37.85                52.0         3.8462
...          ...       ...                 ...            ...
20635    -121.09     39.48                25.0         1.5603
20636    -121.21     39.49                18.0         2.5568
20637    -121.22     39.43                17.0         1.7000
20638    -121.32     39.43                18.0         1.8672
20639    -121.24     39.37                16.0         2.3886

       median_house_value  ocean_proximity_INLAND
ocean_proximity_NEAR BAY  \
0                452600.0                   False
True
1                358500.0                   False
True
2                352100.0                   False
True
3                341300.0                   False
True
4                342200.0                   False
True
...                   ...                     ...
...
20635             78100.0                    True
False
20636             77100.0                    True
False
20637             92300.0                    True
False
20638             84700.0                    True
```

```
False
20639              89400.0                       True
False

       ocean_proximity_NEAR OCEAN  pop_density  house_density
0                           False     2.555556       6.984127
1                           False     2.109842       6.238137
2                           False     2.802260       8.288136
3                           False     2.547945       5.817352
4                           False     2.181467       6.281853
...                           ...          ...            ...
20635                       False     2.560606       5.045455
20636                       False     3.122807       6.114035
20637                       False     2.325635       5.205543
20638                       False     2.123209       5.329513
20639                       False     2.616981       5.254717

[20418 rows x 10 columns]
```

Let's run the model once more to see if the cleaning helped any.

```
cleaned_v3_data = run_model(cleaned_v3)

Mean Squared Error: $2036587664.55
Mean Absolute Error: $29048.12
R^2 Score: 0.8492593892670817
Accuracy within 15.0% tolerance: 62.81%
```

There was again a very slight improvement, but mostly insignifiant. Let's check the feature importances now:

```
columns = cleaned_v3.drop(columns=["median_house_value"]).columns
column_importance = dict(zip(columns, [float(val) for val in
cleaned_v3_data["model"].feature_importances_]))

sorted_col_importance = dict(sorted(column_importance.items(),
key=lambda item: item[1], reverse=True))

sorted_col_importance

{'ocean_proximity_INLAND': 0.5880390405654907,
 'median_income': 0.2376914918422699,
 'pop_density': 0.053230006247758865,
 'ocean_proximity_NEAR OCEAN': 0.02432853728532791,
 'latitude': 0.023902567103505135,
 'housing_median_age': 0.02278636209666729,
 'longitude': 0.02268434502184391,
 'house_density': 0.014687780290842056,
 'ocean_proximity_NEAR BAY': 0.01264976616948843}
```

```python
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

# Define features and target
x = cleaned_v3.drop(columns=["median_house_value"])
y = cleaned_v3["median_house_value"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=419)

# Define the parameter grid
param_grid = {
    "n_estimators": [100, 500, 1000],
    "learning_rate": [0.01, 0.05, 0.1],
    "max_depth": [3, 5, 7],
    "subsample": [0.6, 0.8, 1.0],
    "colsample_bytree": [0.6, 0.8, 1.0]
}

# Initialize the XGBoost regressor
xgb_model = XGBRegressor(objective="reg:squarederror",
random_state=419)

# Set up GridSearchCV
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid,
scoring="neg_mean_squared_error", cv=5, verbose=1)

# Fit the GridSearchCV to find the best parameters
grid_search.fit(X_train, y_train)

# Get the best parameters
print("Best Parameters:", grid_search.best_params_)

# Train the model with the best parameters
best_model = grid_search.best_estimator_

# Predict on the test set
y_pred = best_model.predict(X_test)

# Evaluate the optimized model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Optimized Mean Squared Error: ${mse:.2f}")
print(f"Optimized Mean Absolute Error: ${mae:.2f}")
print(f"Optimized R^2 Score: {r2}")
```

```
# Accuracy within tolerance
tolerance = 0.15
accuracy = ((abs(y_pred - y_test) / y_test) <= tolerance).mean()
print(f"Accuracy within {tolerance * 100}% tolerance: {accuracy:.2%}")
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
Best Parameters: {'colsample_bytree': 0.6, 'learning_rate': 0.05,
'max_depth': 7, 'n_estimators': 1000, 'subsample': 1.0}
Optimized Mean Squared Error: $1843834097.42
Optimized Mean Absolute Error: $27400.77
Optimized R^2 Score: 0.8635262882255974
Accuracy within 15.0% tolerance: 66.26%
```

With this in mind, we now can adjust the `run_model()` function and focus more on data engineering since the improvements are modest. For graphing later, I will store the above results in a dictionary by running the new model on the same dataset, `cleaned_v3`.

```
# From above:
# Best Parameters: {'colsample_bytree': 0.6, 'learning_rate': 0.05,
'max_depth': 7, 'n_estimators': 1000, 'subsample': 1.0}

def run_better_model(df):
    x = df.drop(columns=["median_house_value"])
    y = df["median_house_value"]

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=419)

    # Initialize the XGBoost regressor
    original_model = XGBRegressor(objective="reg:squarederror",
n_estimators=1000, learning_rate=0.05, max_depth=7,
colsample_bytree=0.6, subsample=1.0)

    # Train the model
    original_model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = original_model.predict(X_test)

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"Mean Squared Error: ${mse:.2f}")
    print(f"Mean Absolute Error: ${mae:.2f}")
    print(f"R^2 Score: {r2}")
```

```
    # Medical tolerance (15%)
    tolerance = 0.15

    # Calculate accuracy
    accuracy = ((abs(y_pred - y_test) / y_test) <= tolerance).mean()
    print(f"Accuracy within {tolerance * 100}% tolerance:
{accuracy:.2%}")

    return {"model": original_model, "mae": mae, "mse": mse, "r^2":
r2, "accuracy": accuracy}

better_model_cleaned_v3_data = run_better_model(cleaned_v3)
```

```
Mean Squared Error: $1843834097.42
Mean Absolute Error: $27400.77
R^2 Score: 0.8635262882255974
Accuracy within 15.0% tolerance: 66.26%
```

There does seem to be a cap for the median house value ($500,000), so let's see what happens if all values at that cap is removed (as it may skew the data).

```
df_v4 = cleaned_v3.copy()

# Remove the cap on house value
df_v4 = df_v4[df_v4["median_house_value"] < 500000]

df_v4
```

```
       longitude  latitude  housing_median_age  median_income  \
0        -122.23     37.88                41.0         8.3252
1        -122.22     37.86                21.0         8.3014
2        -122.24     37.85                52.0         7.2574
3        -122.25     37.85                52.0         5.6431
4        -122.25     37.85                52.0         3.8462
...          ...       ...                 ...            ...
20635    -121.09     39.48                25.0         1.5603
20636    -121.21     39.49                18.0         2.5568
20637    -121.22     39.43                17.0         1.7000
20638    -121.32     39.43                18.0         1.8672
20639    -121.24     39.37                16.0         2.3886

       median_house_value  ocean_proximity_INLAND
ocean_proximity_NEAR BAY  \
0                452600.0                   False
True
1                358500.0                   False
True
2                352100.0                   False
True
3                341300.0                   False
```

```
True
4                 342200.0                        False
True
...                    ...                          ...
...
20635              78100.0                         True
False
20636              77100.0                         True
False
20637              92300.0                         True
False
20638              84700.0                         True
False
20639              89400.0                         True
False

       ocean_proximity_NEAR OCEAN  pop_density  house_density
0                           False     2.555556       6.984127
1                           False     2.109842       6.238137
2                           False     2.802260       8.288136
3                           False     2.547945       5.817352
4                           False     2.181467       6.281853
...                           ...          ...            ...
20635                       False     2.560606       5.045455
20636                       False     3.122807       6.114035
20637                       False     2.325635       5.205543
20638                       False     2.123209       5.329513
20639                       False     2.616981       5.254717

[19434 rows x 10 columns]
```

```
v4_data = run_better_model(df_v4)
```

```
Mean Squared Error: $1652208484.28
Mean Absolute Error: $26686.92
R^2 Score: 0.8292834800864093
Accuracy within 15.0% tolerance: 65.35%
```

The accuracy went down a bit, but the mean absolute error is also a bit smaller. This is most likely due to there being a lot of cap-value houses that would have raised the MAE that are no longer in the dataset. It may be important to keep these values instead (and go with `cleaned_v3`).

Before graphing current results, let's try one more model, where we strictly keep the best features. Let's check the feature importance one more time and remove any significantly nonimportant features.

```
columns = cleaned_v3.drop(columns=["median_house_value"]).columns
column_importance = dict(zip(columns, [float(val) for val in
better_model_cleaned_v3_data["model"].feature_importances_]))
```

```
sorted_col_importance = dict(sorted(column_importance.items(),
key=lambda item: item[1], reverse=True))

sorted_col_importance
```

```
{'ocean_proximity_INLAND': 0.5031577348709106,
 'median_income': 0.14498667418956757,
 'latitude': 0.06451987475156784,
 'ocean_proximity_NEAR OCEAN': 0.0592774972319603,
 'pop_density': 0.05639267712831497,
 'house_density': 0.05354450270533562,
 'longitude': 0.05287262424826622,
 'ocean_proximity_NEAR BAY': 0.03992283344268799,
 'housing_median_age': 0.02532562054693699}
```

Using 0.05 as a cutoff, we can remove `ocean_proximity_NEAR BAY` and `housing_median_age`.

```
df_v5 = cleaned_v3.copy()

# Remove any feature with a score of < 0.05
df_v5 = df_v5.drop(columns=["ocean_proximity_NEAR BAY",
"housing_median_age"])

df_v5
```

```
       longitude  latitude  median_income  median_house_value  \
0        -122.23     37.88         8.3252            452600.0
1        -122.22     37.86         8.3014            358500.0
2        -122.24     37.85         7.2574            352100.0
3        -122.25     37.85         5.6431            341300.0
4        -122.25     37.85         3.8462            342200.0
...          ...       ...            ...                 ...
20635    -121.09     39.48         1.5603             78100.0
20636    -121.21     39.49         2.5568             77100.0
20637    -121.22     39.43         1.7000             92300.0
20638    -121.32     39.43         1.8672             84700.0
20639    -121.24     39.37         2.3886             89400.0

       ocean_proximity_INLAND  ocean_proximity_NEAR OCEAN  pop_density
\
0                       False                       False     2.555556

1                       False                       False     2.109842

2                       False                       False     2.802260

3                       False                       False     2.547945
```

```
4                      False                      False    2.181467
...                      ...                        ...         ...
20635                   True                      False    2.560606
20636                   True                      False    3.122807
20637                   True                      False    2.325635
20638                   True                      False    2.123209
20639                   True                      False    2.616981

       house_density
0           6.984127
1           6.238137
2           8.288136
3           5.817352
4           6.281853
...              ...
20635       5.045455
20636       6.114035
20637       5.205543
20638       5.329513
20639       5.254717

[20418 rows x 8 columns]
```

```
v5_data = run_better_model(df_v5)
```

```
Mean Squared Error: $1981654355.69
Mean Absolute Error: $28515.40
R^2 Score: 0.8533253475710927
Accuracy within 15.0% tolerance: 64.57%
```

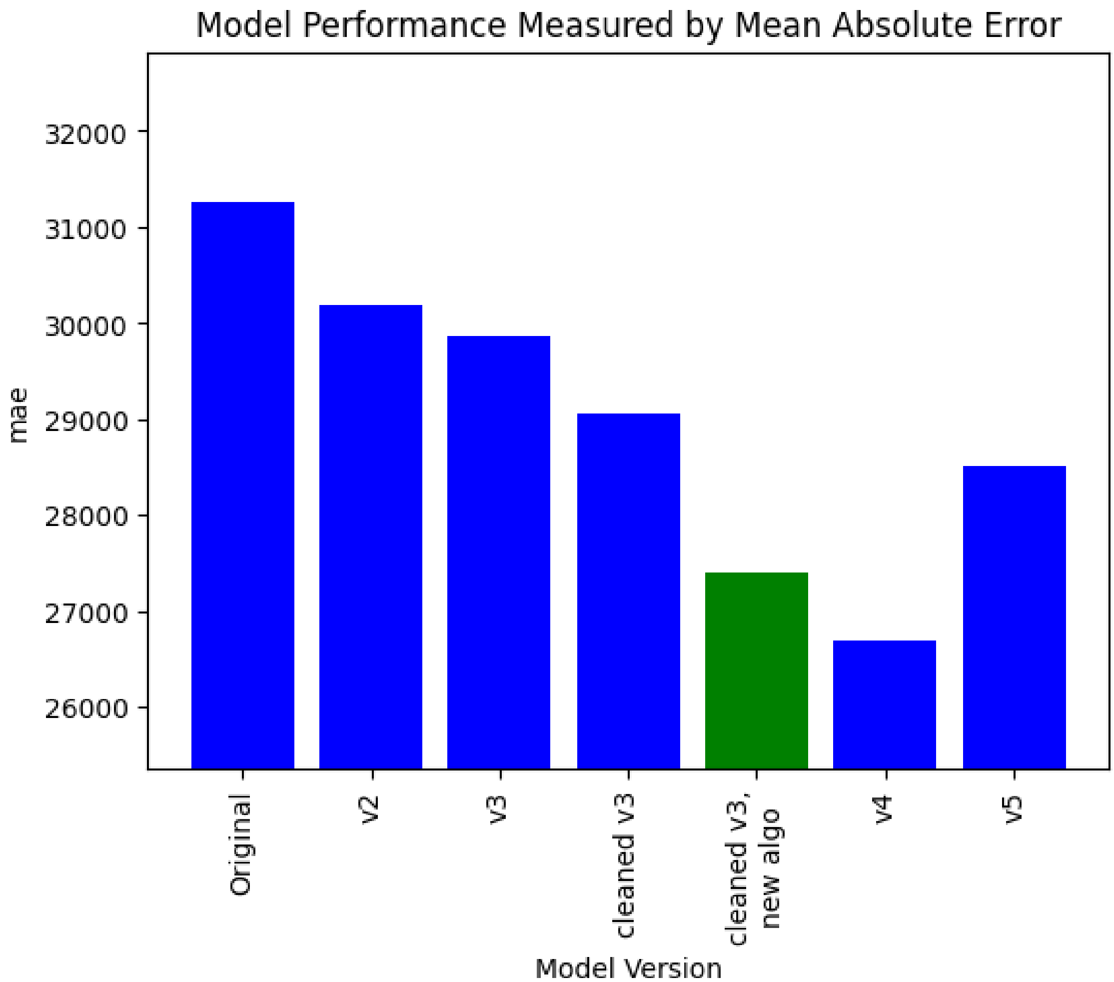
This is still worse than using `cleaned_v3` on the better model.

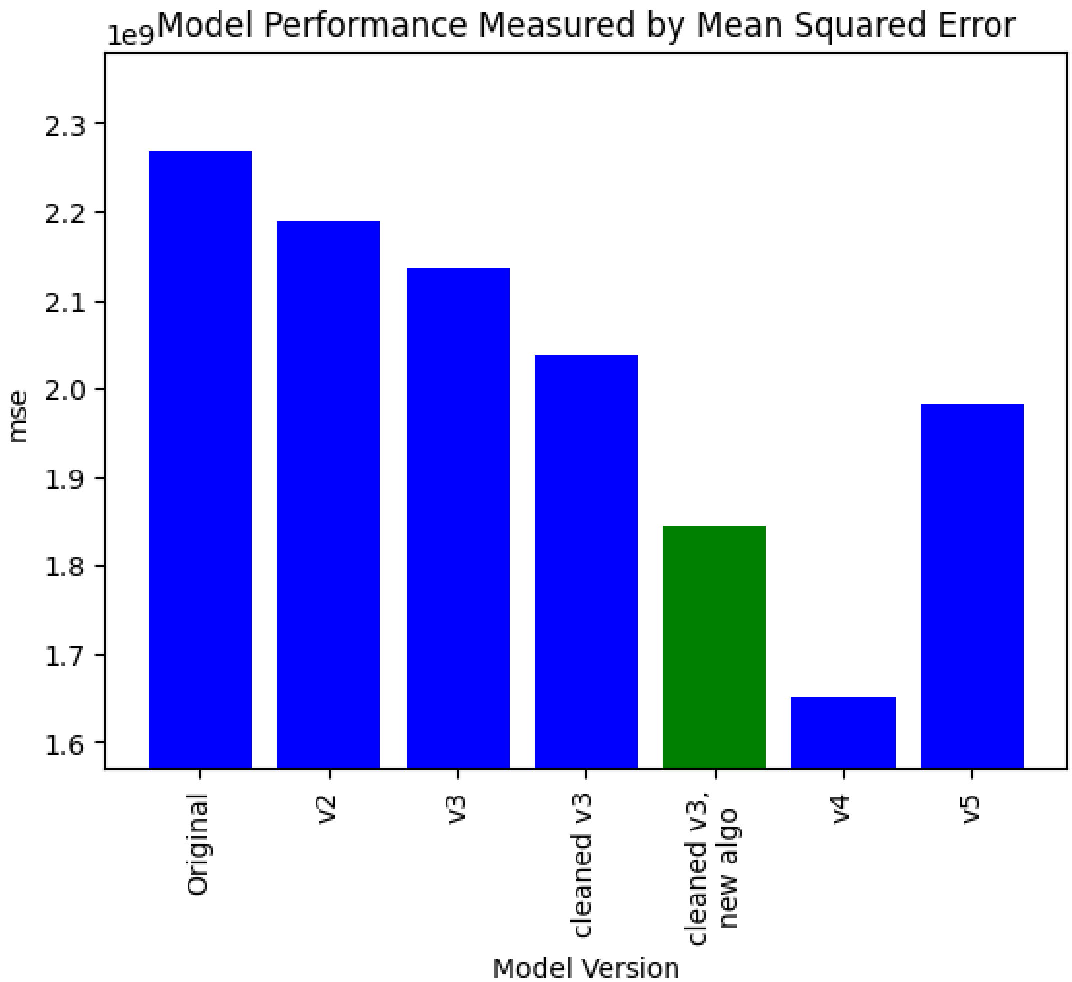
This is a tricky dataset as it is extremely large compared to what I project is to be the one used for the actual capstone project and it has real limitations (the target value has a value cap and is old data).

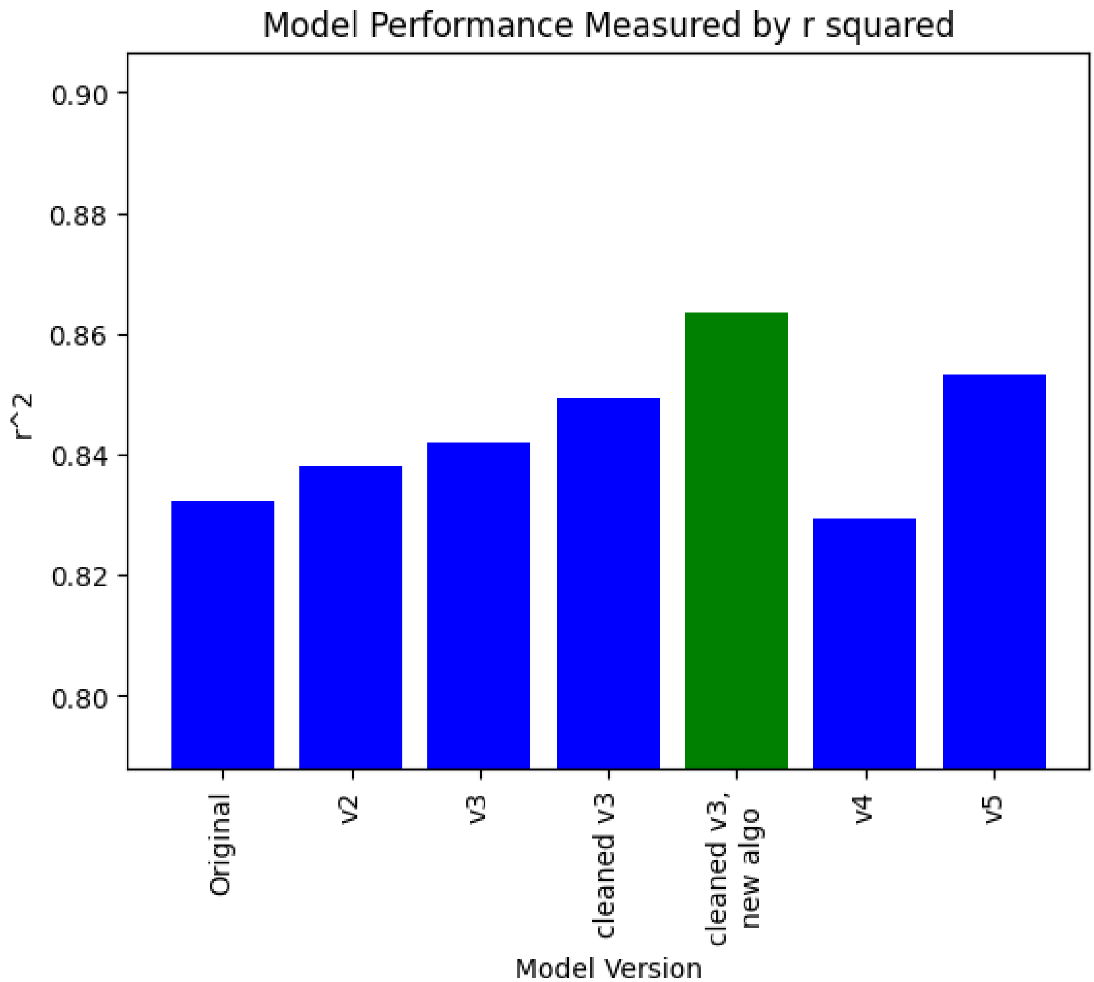
Overall, good improvements were made compared to the original dataset after cleaning, processing, and engineering. The graphs below showcase the models over time.

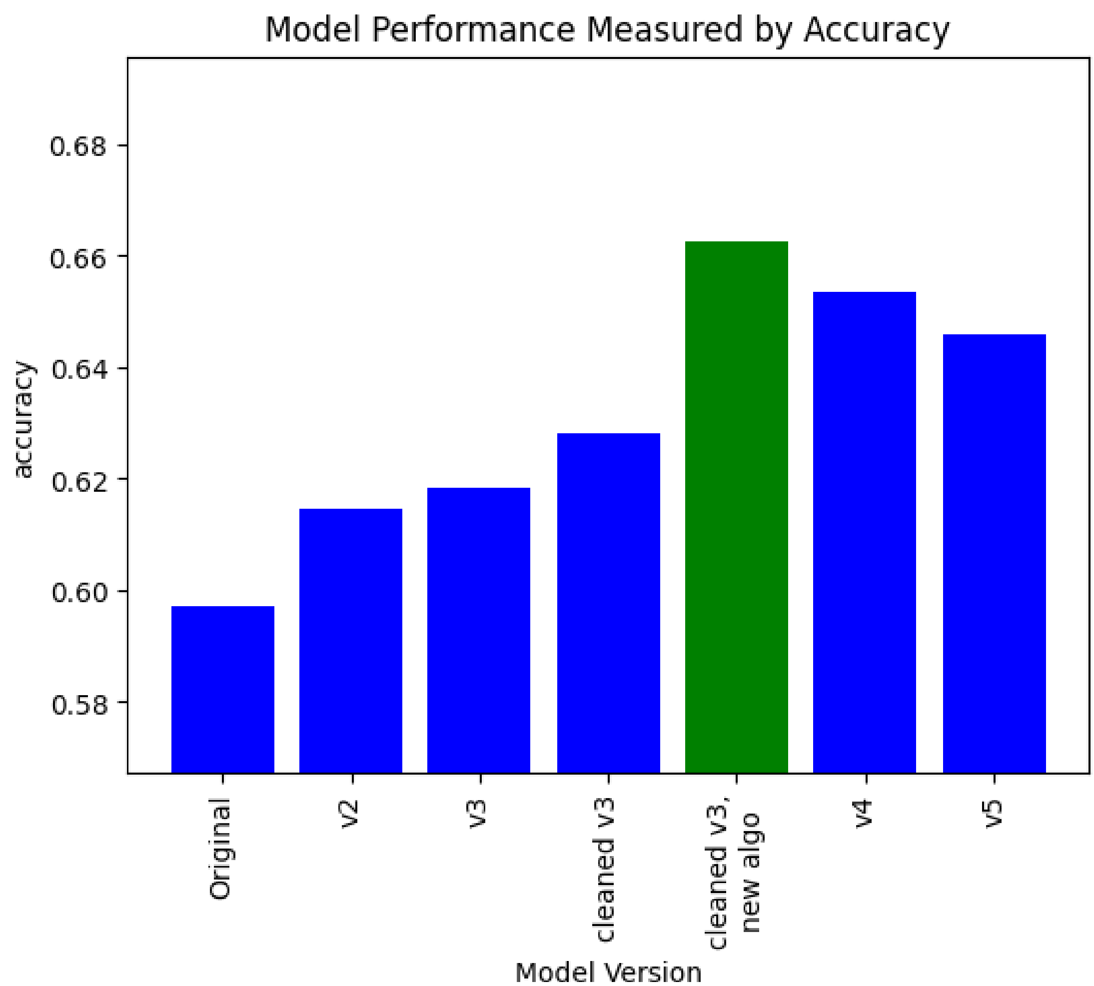
```
import matplotlib.pyplot as plt

all_data = [original_data, v2_data, v3_data, cleaned_v3_data, 
better_model_cleaned_v3_data, v4_data, v5_data]
names = ["Original", "v2", "v3", "cleaned v3", "cleaned v3,\nnew
```

```
algo", "v4", "v5"]
colors = ["blue", "blue", "blue", "blue", "green", "blue", "blue"]

attributes = ["mae", "mse", "r^2", "accuracy"]
full_attributes = ["Mean Absolute Error", "Mean Squared Error", "r
squared", "Accuracy"]

# Make the bar chart
for attribute in attributes:
    # Create a new separate figure
    plt.figure()

    specific_data = []
    # Go through each set of data and collect the correct subset
    for data in all_data:
        specific_data.append(data[attribute])

    # Make the bar chart
    bars = plt.bar(names, specific_data, color=colors)

    # Rotate model names so they fit without overlap
    plt.xticks(rotation=90)

    # Set the y limits to better show the differences
    y_min = min(specific_data) * 0.95
    y_max = max(specific_data) * 1.05
    plt.ylim(y_min, y_max)

    # Labels
    plt.xlabel("Model Version")
    plt.ylabel(attribute)
    plt.title(f"Model Performance Measured by
{full_attributes[attributes.index(attribute)]}")

    plt.show()
```


Model Performance Measured by Mean Absolute Error
mae
32000
31000
30000
29000
28000
27000
26000
Original
v2
v3
cleaned v3
cleaned v3,
new algo
v4
v5
Model Version


Model Performance Measured by Mean Squared Error
1e9
2.3
2.2
2.1
2.0
1.9
1.8
1.7
1.6
mse
Original
v2
v3
cleaned v3
cleaned v3,
new algo
v4
v5
Model Version


Model Performance Measured by r squared
0.90
0.88
0.86
0.84
0.82
0.80
r^2
Original
v2
v3
cleaned v3
cleaned v3,
new algo
v4
v5
Model Version



Below is a `pandas` table representation of the data.

```
data_breakdown = pd.DataFrame()

# Go through every index (every list has the same indices)
for index in range(len(names)):
    # Ensure name is properly formatted
    name = names[index]

    # For the sake of spacing, replace the newline with a regular
space
    if "\n" in name:
        name = name.replace("\n", " ")

    # Format the values so they're more readable
    values = list(all_data[index].values())[1:]

    for value_index in range(len(values)):
        # Make accuracy a percentage
        if value_index == len(values)-1:
```

```
                values[value_index] *= 100

            values[value_index] = f"{values[value_index]:,.2f}"

            # Add percentage sign if on accuracy
            if value_index == len(values)-1:
                values[value_index] += "%"

            # Add a dollar sign if is a money error
            if value_index in (0, 1):
                values[value_index] = "$" + values[value_index]

        # Add the attributes
        data_breakdown[name] = values

# Set the labels for the table
data_breakdown.index = full_attributes

data_breakdown
```

```
                              Original                 v2
v3  \
Mean Absolute Error         $31,261.91         $30,181.74
$29,871.71
Mean Squared Error   $2,267,413,855.57  $2,189,646,288.23
$2,135,919,585.43
r squared                         0.83               0.84
0.84
Accuracy                        59.70%             61.46%
61.83%

                           cleaned v3 cleaned v3, new algo  \
Mean Absolute Error         $29,048.12          $27,400.77
Mean Squared Error   $2,036,587,664.55    $1,843,834,097.42
r squared                         0.85                 0.86
Accuracy                        62.81%               66.26%

                                    v4                 v5
Mean Absolute Error         $26,686.92         $28,515.40
Mean Squared Error   $1,652,208,484.28  $1,981,654,355.69
r squared                         0.83               0.85
Accuracy                        65.35%             64.57%
```