Regional Home Value Prediction Model

Kenneth DeVoe

Steve Amancha

Fernando Calderon

University of San Diego

AAI500 – Probability & Statistics for AI

Instructor: Dallin Munger

June 26, 2023

Introduction

The purpose of this project is to develop a predictive model for estimating the average home value for a zip code. Two datasets are being employed, both focusing on housing values categorized by zip code. These datasets consist of comprehensive lists of regional zip codes, with each zip code represented as an individual row. To streamline the dataset, a selection of the 48 most informative variables was made, which was then used to train the predictive model. This technical report outlines the data cleaning and preparation steps undertaken to ensure the datasets are suitable for analysis, followed by model analysis and selection to make home value predictions.

Data Cleaning/Preparation

The initial step involved filtering the datasets to select the most significant features for analysis. From the "raw_tax_df" dataset, columns pertaining to zip code-related information were extracted. Similarly, the "market_health_df" dataset was filtered to retain columns relevant to market health indicators. This filtering process ensured that only the most relevant features were considered for further analysis. The datasets were then merged based on the common columns "ZIPCODE" in the "raw_tax_df" dataset and "RegionName" in the "market_health_df" dataset. The resulting merged dataset, named "merged_df," contained a combination of information from both datasets, enabling a comprehensive analysis of housing values categorized by zip code.

To prepare for model training and evaluation splits were performed on the merged dataset. First, the merged dataset was split into a training set and a holdout set. 80% of zip codes were randomly selected for training, 10% for validation and 10% for a final test set. The total number of zip codes held in the training, validation and test sets was 11090, 1392 and 1393 respectively. Finally the Zillow Home Value Index or ZHVI for short (Allison, 2022) was selected as the target variable and the remaining features were separated to be used as input.

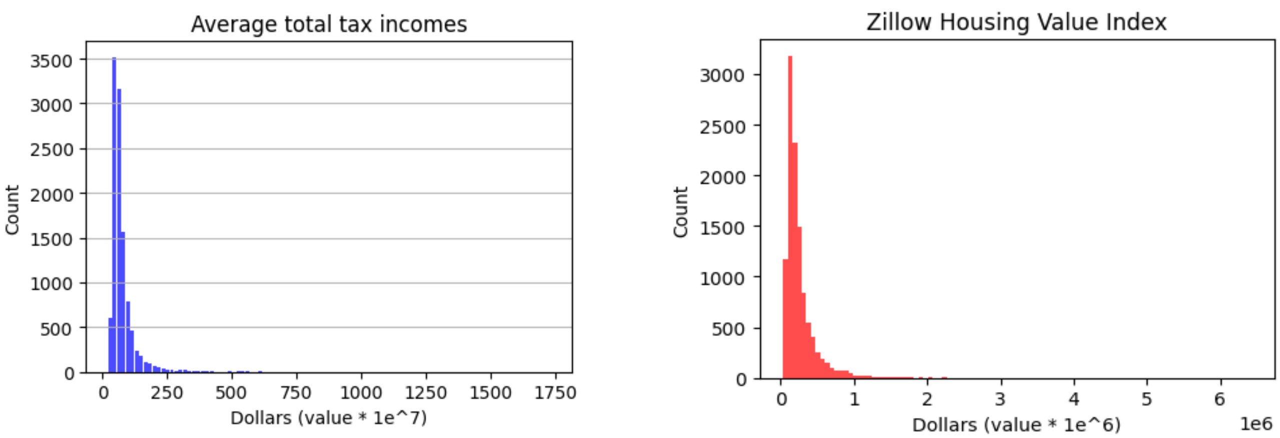
The final steps in data preparation was to check for and impute missing values as well as to normalize the input data. To evaluate the presence of missing values within the input features and best determine the imputation method, the number and percentage of missing values for each feature were calculated. Results showed that five features contained missing values, ranging from 77.4% missing down to only 1.0% missing. Based on these results two inputs (SellForGain, ForeclosureRatio) were dropped due to a high ratio of missing values and three features (NegativeEquity, Delinquency and DaysOnMarket) were imputed as they were missing 2.7% or less of their initial values. To impute the missing values, the K-Nearest Neighbors (KNN) imputation method was employed using the KNNImputer (Htoon, 2020). A parameter value of 5 was chosen to determine the number of neighbors used for imputation.

Exploratory Data Analysis

To gain insights into the dataset, exploratory data analysis (EDA) techniques were applied. First, key features and target values were analyzed to check for data distribution and any discrepancies. From the plot in Figure 1, it can be seen that both the average tax income and ZHVI have a strong skew to the right. This indicates that even though each individual zip code is in turn a collection of the average values, collectively zip codes are still skewed with some particular zip codes showing comparatively high tax incomes and home values. In contrast the Housing Market Health Index (Zillow Research, 2013)value showed a uniform distribution across all zip codes.

Figure 1

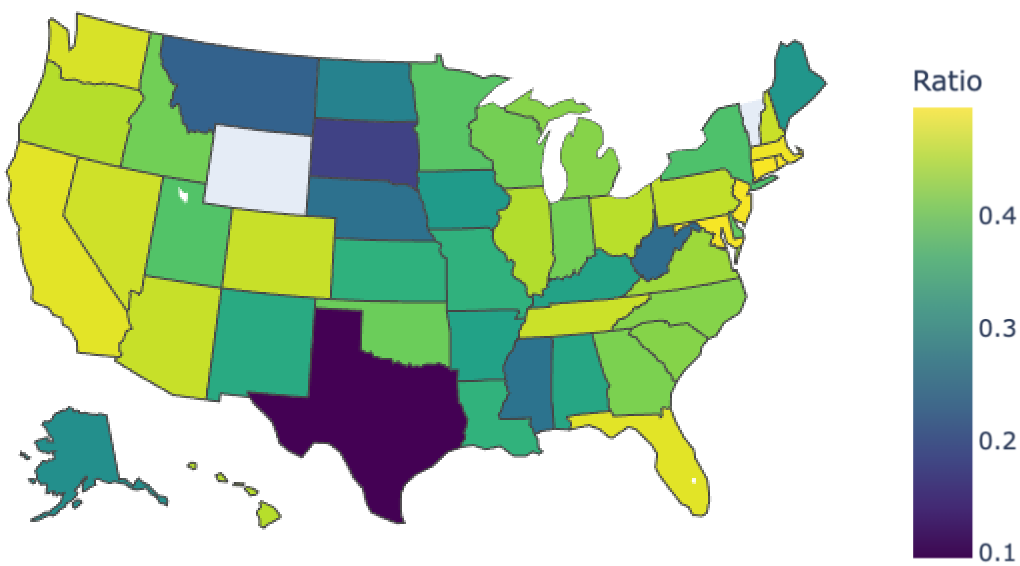
Distribution of key parameters Total Tax Income and Zillow Housing Value Index



One side effect from merging two different datasets together was a reduction in the total amount of available zip codes that were present in both original datasets. While the dataset with tax information contained 27,760 zip codes, the market health dataset contained only 14,089 zip codes. After merging the final dataset contained a subset of the two at 14,003 zip codes total. To analyze if this drop in zip codes was biased in any particular region the ratio of drop in zip codes was plotted per state in Figure 2 . In general there does not appear to be significant regional trends for the data loss and it appears to be random by state. Some states have bigger losses than others such as Texas, and this should be taken into consideration when predicting values in those regions.

Figure 2

Data loss ratio by state from merging datasets across zip codes



Note. *Darker regions indicate more data loss and are less represented in the dataset.*

Model Selection

Four different models were compared in terms of capability to predict home values for a given zip code. Two models were based on a standard ordinary least squares regression model, using the Statsmodels library's Ordinary Least Squares (OLS) method with either a gaussian or gamma link function to make predictions. One model was created as a simple neural network consisting of two fully connected layers implemented in the pytorch framework. The final model was a simple implementation of the XGBoost algorithm for value prediction.

The model's performance was evaluated by generating predictions on the validation set and comparing them against the actual target values. The root mean squared error (RMSE) was computed as the main quantitative measure of the model's accuracy. In addition the maximum difference from a single prediction to actual value was also calculated to represent the worst case of the model within the validation dataset. To compare models against each other the overall RMSE for the validation set was used with the max difference and the model with the lowest error was selected. Table 1 shows the RMSE values for the different models.

Table 1

Comparison of model performance using RMSE and maximum difference

Model	**RMSE ($)**	**Max Difference ($)**
GLM Gaussian	88,221	763,736
GLM Gamma	96,931	983,295
Neural Network	80,627	926,255
XGBoost	62,115	648,382

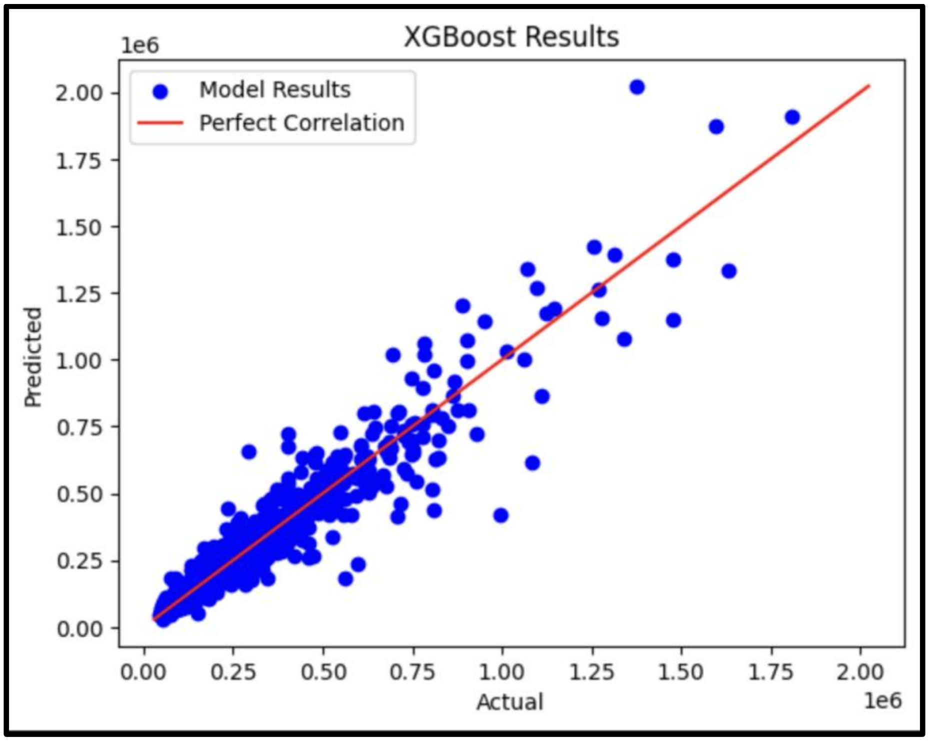
As it can be seen from Table 1 the XGBoost gave the best overall performance in terms of general RMSE as well as the lowest maximum error. XGBoost was selected as the model to use for housing value predictions based on these results.

Model Analysis

The overall performance of the XGBoost model is shown in Figure 3.

Figure 3

XGBoost value predictions versus ideal values



Note. *The red line represents ideal predictions whereas blue dots are predicted values from XGBoost.*

The actual home values by zip code are represented in the x-axis and the models prediction is given on the y-axis. The red line represents an ideal prediction that exactly matches the actual home values. As can be seen from Figure 3 prediction values tend to cluster closer to the ideal line at lower home values and spread out more for higher home values. This trend is confirmed by Table 2 showing the RMSE and max difference by $100k interval amounts.

Table 2

XGBoost performance by home value interval

	Interval	Count	RMSE	Max_Difference
0	$0.0k to $100.0k	169	21462.3	106746.7
1	$100.0k to $200.0k	575	24365.9	127943.8
2	$200.0k to $300.0k	315	42257.2	365805.9
3	$300.0k to $400.0k	131	54280.0	160963.2
4	$400.0k to $500.0k	73	91172.6	321746.4
5	$500.0k to $600.0k	38	113522.7	380006.2
6	$600.0k to $700.0k	30	100812.1	327768.2
7	$700.0k to $800.0k	22	146740.0	295872.4
8	$800.0k to $900.0k	14	178449.7	370654.9
9	$900.0k to Maximum	25	259402.3	648382.9
10	All Data	1392	62115.9	648382.9

The RMSE error in dollars tends to increase with increasing home values. There are a few potential causes for this increase. The first is that as a percentage of the actual home value the error inherently increases in dollar value. The second is that higher values tend to have less examples overall and less opportunity for the model to train. Finally it may be that predicting home values in high value regions may be more difficult than lower value regions using the inputs available to the model.

The overall performance of XGBoost appears to match well with actual values indicating it can be very useful in predicting regional home values. In particular the error in dollars is lowest in home values under $400k which represents the majority of zip codes in the US. Model performance for higher value zip codes could potentially be improved by further model tuning or gathering more data from high home value regions for model training.

Conclusion and Recommendations

The development of a predictive model for estimating home values based on zip codes necessitates careful consideration of data sources and feature selection. In this project, the Market Health Index dataset and the 2017 individual tax income dataset are utilized to train a linear regression model. The objective was to achieve accurate predictions of home values by identifying key variables from the tax dataset that demonstrate strong correlations with the corresponding Zillow index values.

The selection of datasets and the process of feature selection were critical elements in ensuring the effectiveness of the model. By focusing on variables related to market health and income, the aim was to capture fundamental factors that impact housing values. The datasets were filtered to extract the most pertinent information for the predictive model.

Of the four models evaluated for home value prediction XGBoost gave the best overall performance by minimizing the difference between home predicted values and actual values. A few items should be taken into consideration when using the model such as different performance at different home values and potentially different performance in regions across the US. As this model was developed on data across the US one potential area for future study and improvement would be to specifically train or fine tune the model on regions of interest.

References

Allison, M. (2022) ZHVI User Guide, https://www.zillow.com/research/zhvi-user-guide/

Bemporad, A. (2023). A Piecewise Linear Regression and Classification Algorithm With Application to Learning and Model Predictive Control of Hybrid Systems. IEEE Transactions on Automatic Control, Automatic Control, IEEE Transactions on, IEEE Trans. Automat. Contr, 68(6), 3194–3209. https://doi-org.sandiego.idm.oclc.org/10.1109/TAC.2022.3183036

Cross Sihombing, D. J., Othernima, D. C., Manurung, J., & Sagala, J. R. (2023). Comparative Models of Price Estimation Using Multiple Linear Regression and Random Forest Methods. 2023 International Conference on Computer Science, Information Technology and Engineering (ICCoSITE), Computer Science, Information Technology and Engineering (ICCoSITE), 2023 International Conference On, 478–483. https://doi-org.sandiego.idm.oclc.org/10.1109/ICCoSITE57641.2023.10127705

Htoon, K (2020) A Guide to KNN Imputation, Retrieved from: https://medium.com/@kyawsawhtoon/a-guide-to-knn-imputation-95e2dc496e

Kim, S., Wimmer, H., & Kim, J. (2022). Analysis of Deep Learning Libraries: Keras, PyTorch, and MXnet. 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), Software Engineering Research, Management and Applications (SERA), 2022 IEEE/ACIS 20th International Conference On, 54–62. https://doi-org.sandiego.idm.oclc.org/10.1109/SERA54885.2022.9806734

Shajalal, M., Hajek, P., & Abedin, M. Z. (2023). Product backorder prediction using deep neural network on imbalanced data. International Journal of Production Research, 61(1), 302–319. https://doi-org.sandiego.idm.oclc.org/10.1080/00207543.2021.1901153

Dataset 1 - MarketHealthIndex.csv Retrieved from: https://data.world/zillow-data/market-health-index

Dataset 2 - 17zpallnoagi.csv Retrieved from: https://data.world/ian/2013-zip-code-income/workspace/project-summary?agentid=ian&datasetid=2013-zip-code-income

Zillow Research (2023). Zillow Home Value and Sales Forecast. https://www.zillow.com/research/may-2023-home-value-sales-forecast-32643/

Appendix

Library and Data Imports

In [ ]:
```
import scipy.stats as stats
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm

import torch
import torch.nn as nn
import torch.nn.functional as F
from tqdm import tqdm
```

In [ ]:
```
# read local csv files and put into data frames
data_url = '17zpallnoagi.csv'
zpallnoagi_csv = pd.read_csv(data_url)
raw_tax_df = pd.DataFrame(zpallnoagi_csv)
mhi_data_url = 'MarketHealthIndex_Zip.csv'
market_health_csv = pd.read_csv(mhi_data_url, on_bad_lines='skip', encoding = '
market_health_df = pd.DataFrame(market_health_csv)
```

Data Cleaning and Imputation

Downselection of Features

In [ ]:
```
# Columns used for both data sets
market_health_cols = ['RegionName','MarketHealthIndex','SellForGain','Foreclosu
                      'DaysOnMarket','ZHVI']

zip_tax_cols = ['N1','ZIPCODE','MARS1','MARS2','MARS4','NUMDEP','A00100','N0265
                'A01000','A01700','SCHF','A02300','A02500','N26270','N03220','A
                'A17000','A18425','A18500','A19300','N19570','A19700','A20950',
                'A07180','N07220','A07220','N09400','A09400','A10600','N11070',

# filtered and merged together raw data sets
tax_zip_df = raw_tax_df.filter(zip_tax_cols)
mh_df = market_health_df.filter(market_health_cols, axis=1)
merged_df = pd.merge(tax_zip_df, mh_df, left_on="ZIPCODE", right_on="RegionName
grouped_df = merged_df.drop(columns=['RegionName'])

# Print the length of the tax_zip_df and mh_df
print("Length of tax_zip_df: ", len(tax_zip_df))
print("Length of mh_df: ", len(mh_df))
print("Length of merged_df: ", len(merged_df))
```

```
# most simpliest data frame with zipcode as the key and all other columns as fe
display(grouped_df.head())
```

```
Length of tax_zip_df:  27760
Length of mh_df:  14089
Length of merged_df:  14003
```

	N1	ZIPCODE	MARS1	MARS2	MARS4	NUMDEP	A00100	N02650	A02650	A0020
0	5130.0	35004	2140.0	2120.0	780.0	3350.0	289966.0	5130.0	292671.0	236776
1	3170.0	35005	1350.0	870.0	900.0	2230.0	124916.0	3170.0	125810.0	102620
2	1210.0	35006	440.0	580.0	170.0	820.0	59411.0	1210.0	59725.0	46012
3	11930.0	35007	4720.0	5180.0	1790.0	8840.0	706211.0	11930.0	714402.0	555765
4	7890.0	35010	3000.0	2710.0	2060.0	5850.0	387333.0	7890.0	391523.0	262452

5 rows × 52 columns

Splitting Training and Holdout Data

Only training data will be used for Exploratory Data Analysis

In [ ]:
```
# splitting data
# df_eda is 80% of random data, df_holdout is 20% of the remaining data
df_eda, df_holdout = train_test_split(grouped_df, test_size=0.20, random_state=
display(df_eda.head())
```

	N1	ZIPCODE	MARS1	MARS2	MARS4	NUMDEP	A00100	N02650	A02650
9140	9940.0	13850	4710.0	4340.0	710.0	5100.0	806571.0	9940.0	818461.0
5412	3550.0	20616	1690.0	920.0	800.0	2500.0	238866.0	3550.0	239927.0
5885	3480.0	1566	1640.0	1540.0	240.0	1790.0	315416.0	3480.0	318920.0
4476	13410.0	46815	6620.0	4850.0	1710.0	7980.0	659101.0	13410.0	666299.0
12991	24430.0	23223	13650.0	3550.0	6570.0	14020.0	1010689.0	24430.0	1021980.0

5 rows × 52 columns

Normalizing by Count

In [ ]:
```
n1_url = 'Columns_Used.csv'
Columns_Used = pd.read_csv(n1_url)
Columns_Used[Columns_Used['Description'].str.contains("Number of")]
column_list = list(Columns_Used.Code)
column_list = [x.strip() for x in column_list]

for column in column_list:
    if column != "N1" and column in list(df_eda.columns):
```

```
        df_eda[column] = df_eda[column].div(df_eda["N1"])
```

Imputation of Missing Values

In [ ]:
```
# First remove rows missing the target variable of ZHVI
df_eda = df_eda[df_eda['ZHVI'].notna()]

# Split the data into input and target
train_X = df_eda.drop(columns=['ZHVI'])
train_y = df_eda['ZHVI']

# Calculating the missing values
missing_values = train_X.isnull().sum()
missing_values = missing_values[missing_values > 0]

# Creating datafram to display missing data
cols = ['Feature', 'Number Missing', 'Percent Missing']
missing_df = pd.DataFrame(columns=cols)
for col in missing_values.index:
    missing_df.loc[len(missing_df.index)] = {'Feature': col,
                                    'Number Missing': missing_values[col],
                                    'Percent Missing': missing_values[col] / le

pd.set_option('display.float_format', '{:.1f}'.format)
display(missing_df)

# Remove SellForGain, ForeclosureRatio columns
#train_X = train_X.drop(columns=['SellForGain', 'ForeclosureRatio'])

# Impute missing values using KNN
imputer = KNNImputer(n_neighbors=5)
imputed_values = imputer.fit_transform(train_X)

# Convert the numpy array back into a dataframe
train_X = pd.DataFrame(imputed_values, columns=train_X.columns)

missing_values_after = train_X.isnull().sum()
print('\nTotal missing values after imputation:')
print(len(missing_values_after[missing_values_after > 0]))
```

	Feature	Number Missing	Percent Missing
0	SellForGain	3645	32.9
1	ForeclosureRatio	8581	77.4
2	NegativeEquity	295	2.7
3	Delinquency	295	2.7
4	DaysOnMarket	107	1.0

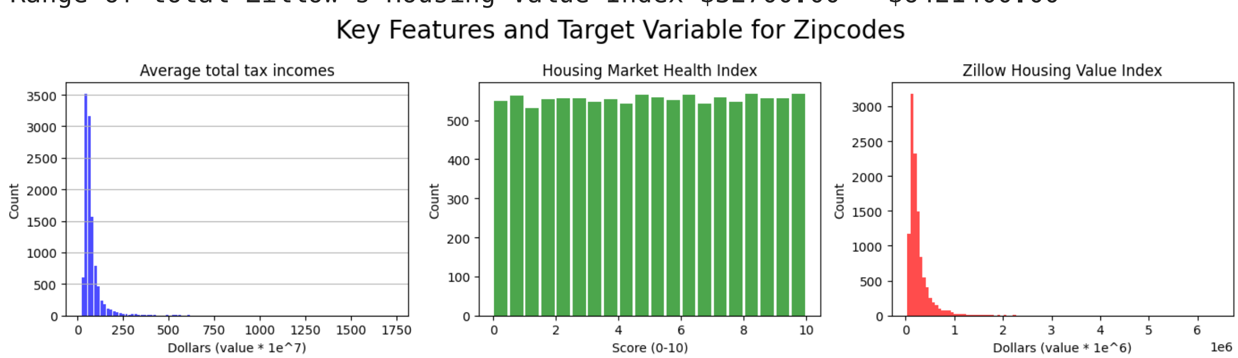
```
Total missing values after imputation:
0
```

Exploratory Data Analysis

Distribution of Key Features and Target Variable

In [ ]:
```
fig, axs = plt.subplots(1,3, figsize=(14,4))

axs[0].ticklabel_format(axis='both')

print(f"Range of total tax income ${min(df_eda['A02650']):.2f} - ${max(df_eda['
# Create a histogram for 'income'
axs[0].hist(df_eda['A02650'], bins=100, color='blue', alpha=0.7, rwidth=0.85)
axs[0].grid(axis='y', alpha=0.9)
axs[0].set_title('Average total tax incomes')
axs[0].set_xlabel('Dollars (value * 1e^7)')
axs[0].set_ylabel('Count')

# Create a histogram for 'market_health_index'
axs[1].hist(df_eda['MarketHealthIndex'], bins=20, color='green', alpha=0.7, rwi
axs[1].set_title('Housing Market Health Index')
axs[1].set_xlabel('Score (0-10)')
axs[1].set_ylabel('Count')

print(f"Range of total Zillow's Housing Value Index ${min(df_eda['ZHVI']):.2f}
# Create a histogram for 'zillow_housing_value_index'
axs[2].hist(df_eda['ZHVI'], bins=100, color='red', alpha=0.7)
axs[2].set_title('Zillow Housing Value Index')
axs[2].set_xlabel("Dollars (value * 1e^6)")
axs[2].set_ylabel("Count")

fig.suptitle('Key Features and Target Variable for Zipcodes', fontsize=20)

plt.tight_layout()
```

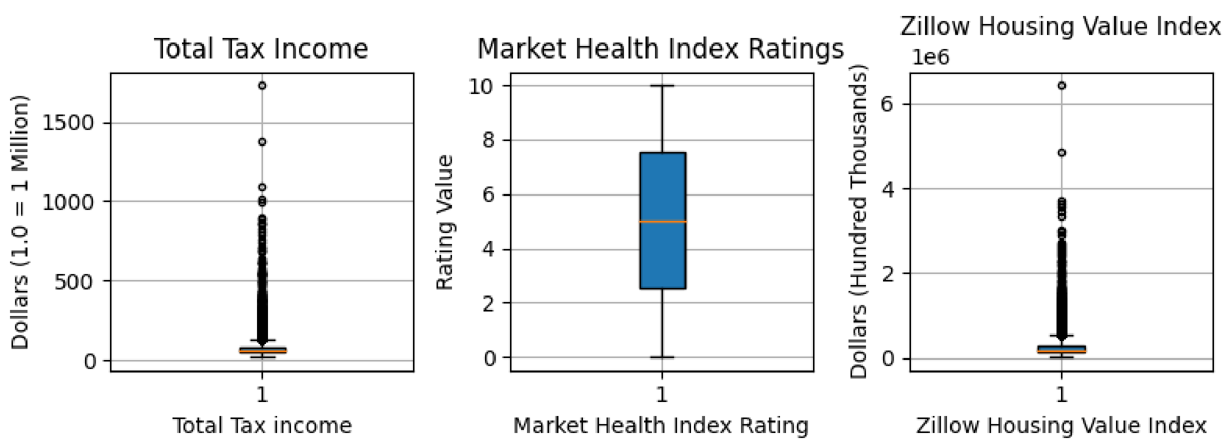
```
Range of total tax income $20.35 - $1730.42
Range of total Zillow's Housing Value Index $32700.00 - $6421400.00
```



Viewing Boxplots

In [ ]:
```
# Showing quartile ranges on total tax income
fig, (ax1,ax2, ax3) = plt.subplots(1, 3, figsize=(8,3))
income_mean = np.mean(df_eda['A02650'].values)
ax1.boxplot(df_eda['A02650'], sym='.', patch_artist=True)
# ax1.axhline(y = income_mean, color = 'r', linestyle = '-')
```

```
ax1.grid(True)
ax1.set_title("Total Tax Income")
ax1.set_xlabel("Total Tax income")
ax1.set_ylabel("Dollars (1.0 = 1 Million)")

# Plotting and labeling MarketHealthIndex boxplot graph
ax2.boxplot(df_eda['MarketHealthIndex'], sym='.', patch_artist=True)
ax2.grid(True)
ax2.set_title("Market Health Index Ratings")
ax2.set_xlabel("Market Health Index Rating")
ax2.set_ylabel("Rating Value")

# Plotting and labeling Zillow Housing Value Index boxplot
ax3.boxplot(df_eda['ZHVI'], sym='.', patch_artist=True)
ax3.grid(True)
ax3.set_title("Zillow Housing Value Index", fontsize="11")
ax3.set_xlabel("Zillow Housing Value Index")
ax3.set_ylabel("Dollars (Hundred Thousands)")

plt.tight_layout()
```



Data Loss from Dataset Merge

In [ ]:
```
available_returns = raw_tax_df.groupby(['STATE']).sum().reset_index()
market_health_zips = market_health_df['RegionName'].tolist()


used_returns = raw_tax_df[raw_tax_df['ZIPCODE'].isin(market_health_zips)]

used_returns = used_returns.groupby(['STATE']).sum().reset_index()


merged_df = pd.merge(available_returns, used_returns, on='STATE', how='inner')

merged_df['Ratio'] = merged_df['N1_y'] / merged_df['N1_x']

import plotly.express as px

# May need pip install --upgrade nbformat

fig = px.choropleth(merged_df, locations="STATE", color="Ratio", hover_name="ST
```

```
                    title='Percentage Use of Zip Codes by State', locationmode=
fig.show()
```

Model Selection

Final Data Preparation

Split holdout into validation and train sets and perform same data cleaning as training values. Normalize all inputs.

In [ ]:
```
# Remove rows missing the target variable of ZHVI
df_holdout = df_holdout[df_holdout['ZHVI'].notna()]

# Split holdout data into validation and test set
val, test = train_test_split(df_holdout, test_size=0.50, random_state=22)

# Split the data into input and target
val_X = val.drop(columns=['ZHVI'])
val_y = val['ZHVI']

test_X = test.drop(columns=['ZHVI'])
test_y = test['ZHVI']

val_imputed_values = imputer.fit_transform(val_X)
test_imputed_values = imputer.fit_transform(test_X)

# Convert the numpy array back into a dataframe
val_X = pd.DataFrame(val_imputed_values, columns=val_X.columns)
test_X = pd.DataFrame(test_imputed_values, columns=test_X.columns)

# Divide by N1 value
n1_url = 'Columns_Used.csv'
Columns_Used = pd.read_csv(n1_url)
Columns_Used[Columns_Used['Description'].str.contains("Number of")]
column_list = list(Columns_Used.Code)
column_list = [x.strip() for x in column_list]

for column in column_list:
    if column != "N1" and column in list(val_X.columns):
        val_X[column] = val_X[column].div(val_X["N1"])
        test_X[column] = test_X[column].div(test_X["N1"])

# Calculating the missing values
missing_values = test_X.isnull().sum()
missing_values = missing_values[missing_values > 0]

# Creating datafram to display missing data
cols = ['Feature', 'Number Missing', 'Percent Missing']
missing_df = pd.DataFrame(columns=cols)
for col in missing_values.index:
    missing_df.loc[len(missing_df.index)] = {'Feature': col,
                                             'Number Missing': missing_values[col],
                                             'Percent Missing': missing_values[col] / le
```

```
pd.set_option('display.float_format', '{:.1f}'.format)
display(missing_df)

# Scale with MinMaxScaler
min_max_scaler = MinMaxScaler()
min_max_scaler.fit(train_X)
train_X_min_max = min_max_scaler.transform(train_X)
val_X_min_max = min_max_scaler.transform(val_X)
test_X_min_max = min_max_scaler.transform(test_X)


# Normalize the train_X data
scaler = StandardScaler()
scaler.fit(train_X)
train_X = scaler.transform(train_X)

# Normalize the val_X and test_X data
val_X = scaler.transform(val_X.to_numpy())
test_X = scaler.transform(test_X.to_numpy())
```

Feature	Number Missing	Percent Missing

```
/Users/kdevoe/Documents/CS/Masters/AAI500/FInal_Project/aai-500-final-group-4/
env/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning:

X does not have valid feature names, but StandardScaler was fitted with featur
e names

/Users/kdevoe/Documents/CS/Masters/AAI500/FInal_Project/aai-500-final-group-4/
env/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning:

X does not have valid feature names, but StandardScaler was fitted with featur
e names
```

Function to analzye results of a given regression model for price prediction.

In [ ]:
```
def analyze_results(actual, predictions, name=""):

    interval_size = 100
    last_interval = 1000
    num_intervals = int(last_interval / interval_size)

    results_df = pd.DataFrame(columns=['Interval', 'Count', 'RMSE', 'Max_Differ

    # Process each interval for value
    for i in range(0, num_intervals):

        filtered_actual = []
        filtered_predictions = []

        # Filter the actual and predictions to the current interval
        lower_bound = interval_size * i * 1000

        if i == num_intervals - 1:
            upper_bound = np.inf
```

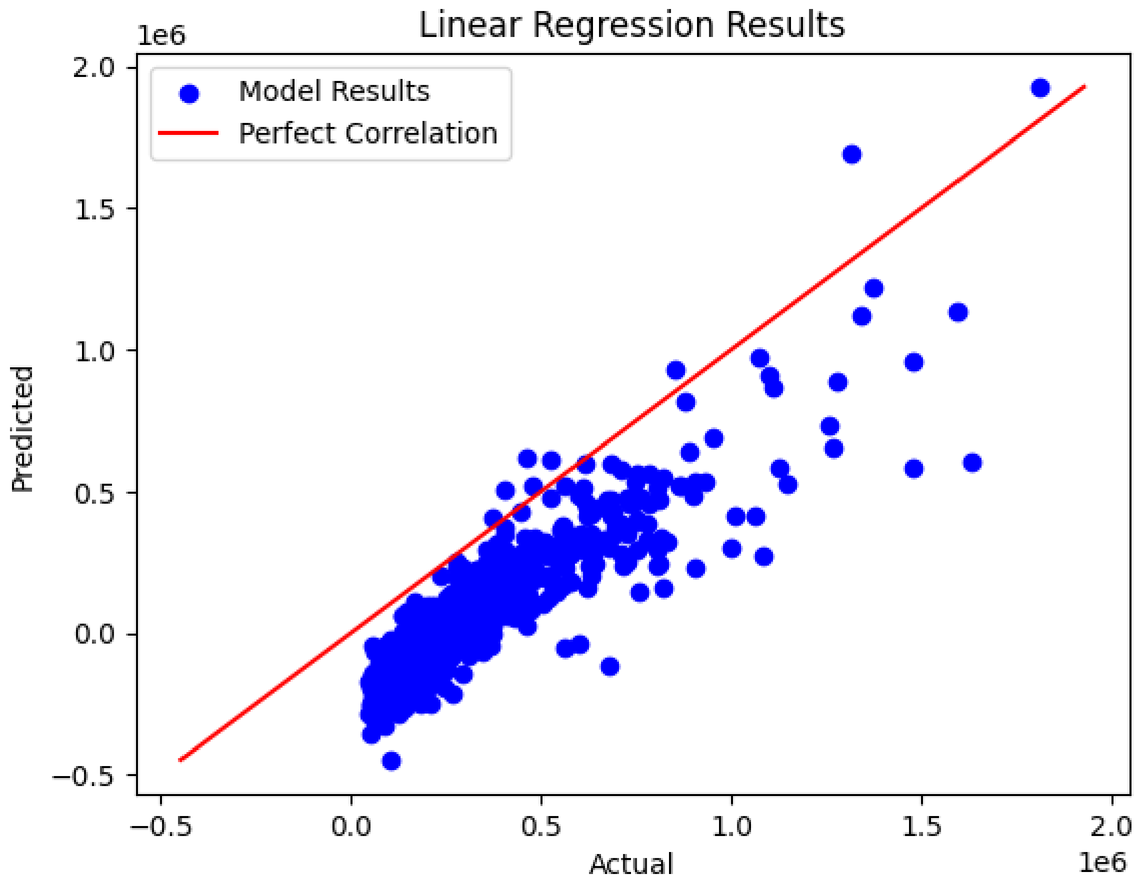
```
        else:
            upper_bound = interval_size * (i + 1) * 1000

        for j in range(len(actual)):
            if actual[j] >= lower_bound and actual[j] < upper_bound:
                filtered_actual.append(actual[j])
                filtered_predictions.append(predictions[j])

        # Calculate the metrics for the interval
        rmse = np.sqrt(mean_squared_error(filtered_actual, filtered_predictions
        max_diff = 0
        for j in range(len(filtered_actual)):
            max_diff = max(abs(filtered_actual[j] - filtered_predictions[j]), m

        lower = f'${lower_bound / 1000}k'
        if upper_bound == np.inf:
            upper = 'Maximum'
        else:
            upper = f'${upper_bound / 1000}k'
        # Store interval results
        results_df.loc[len(results_df.index)] = {'Interval': f'{lower} to {uppe
                                                  'Count': len(filtered_actual)
                                                  'RMSE': rmse,
                                                  'Max_Difference': max_diff}

    # Add the final row for all data
    rmse = np.sqrt(mean_squared_error(actual, predictions))
    max_diff = 0
    for j in range(len(actual)):
        max_diff = max(abs(actual[j] - predictions[j]), max_diff)
    results_df.loc[len(results_df.index)] = {'Interval': 'All Data',
                                              'Count': len(actual),
                                              'RMSE': rmse,
                                              'Max_Difference': max_diff}

    # Plot the results
    plt.scatter(actual, predictions, color='blue')
    plt.title(f'{name} Results')
    plt.xlabel('Actual')
    plt.ylabel('Predicted')

    # Add a line for perfect correlation
    min_value = min(min(actual), min(predictions))
    max_value = max(max(actual), max(predictions))
    plt.plot([min_value, max_value], [min_value, max_value], color='red')
    # Add legend
    plt.legend(['Model Results', 'Perfect Correlation'])

    plt.show()

    return results_df
```

Linear Regression

In [ ]:

```
# Running linear regression with a constant on test set
model = sm.OLS(train_y.values, train_X)
```

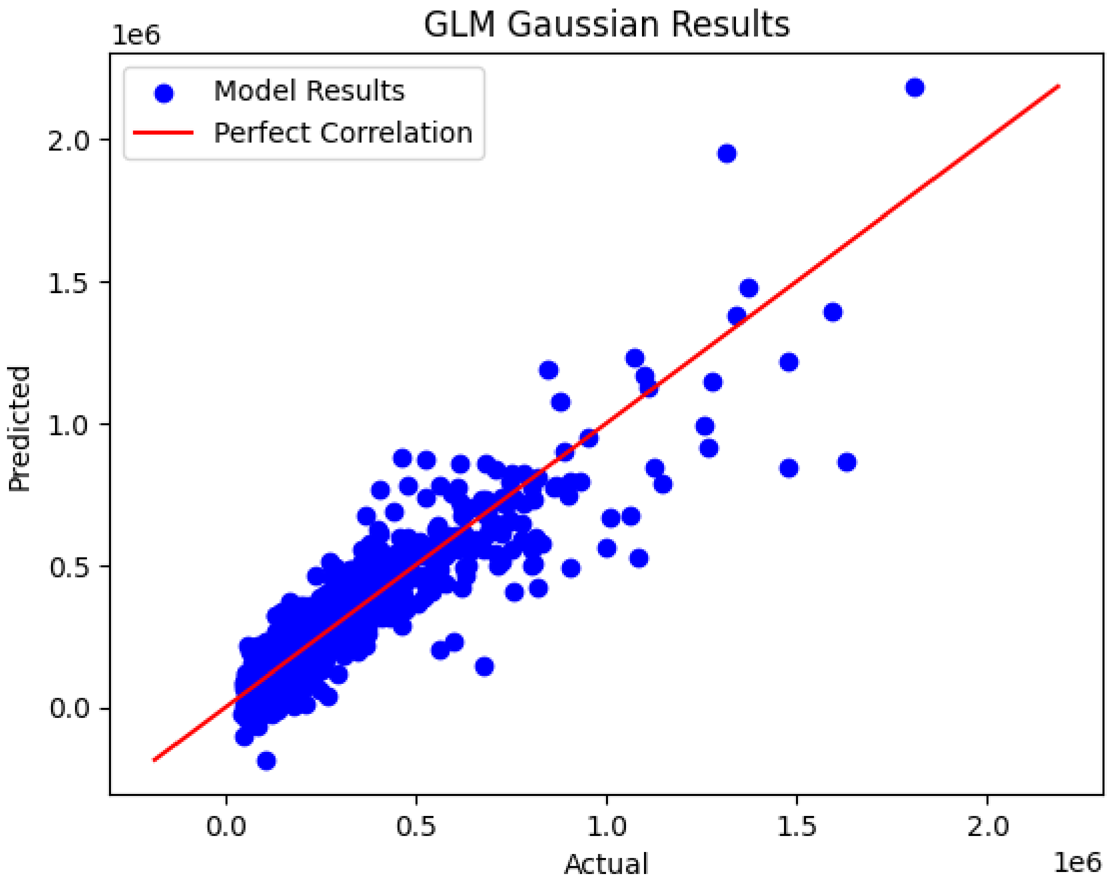
```
results = model.fit()

# Using model to predict on our validation set and to compare predict values
predict_val = results.predict(val_X)

display(analyze_results(val_y.values, predict_val, "Linear Regression"))
```



	Interval	Count	RMSE	Max_Difference
0	0.0kto100.0k	169	271913.5	412205.5
1	100.0kto200.0k	575	261567.2	553146.8
2	200.0kto300.0k	315	250781.2	486521.1
3	300.0kto400.0k	131	247165.8	409475.7
4	400.0kto500.0k	73	249220.0	436186.6
5	500.0kto600.0k	38	311030.6	632751.2
6	600.0kto700.0k	30	318626.7	792107.6
7	700.0kto800.0k	22	362389.2	609121.6
8	800.0kto900.0k	14	414237.8	659050.4
9	$900.0k to Maximum	25	532037.2	1025326.9
10	All Data	1392	272379.0	1025326.9

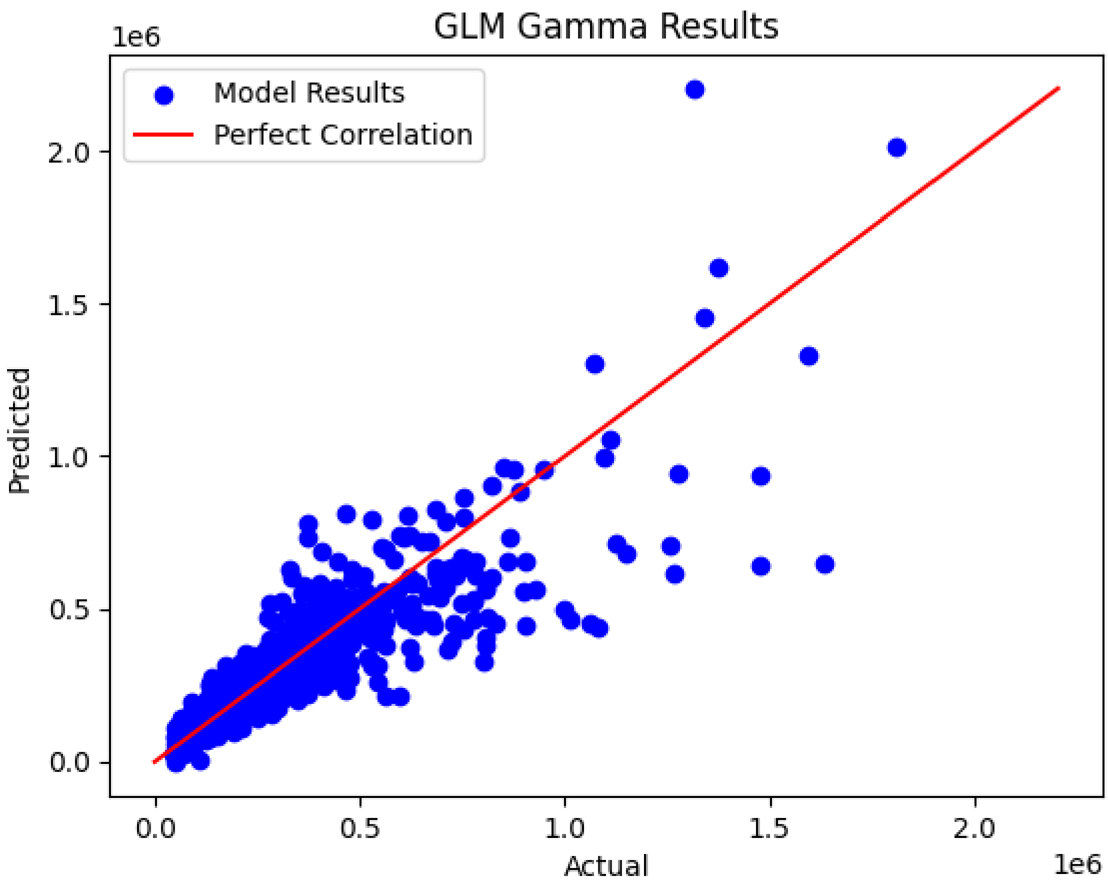
Generalized Linear Models

In [ ]:
```
# Instantiate a Gaussian family model with the default link function.
# Testing test set
glm_gaus = sm.GLM(train_y.values, train_X_min_max, family=sm.families.Gaussian(
res_gaus = glm_gaus.fit()
predict_glm_train = res_gaus.predict()
MSE = mean_squared_error(train_y, predict_glm_train)
sMSE_gaus_train = np.sqrt(MSE)


# Testing on Validation set
predict_glm_val = res_gaus.predict(val_X_min_max)
MSE = mean_squared_error(val_y, predict_glm_val)
sMSE_gaus_val = np.sqrt(MSE)

display(analyze_results(val_y.values, predict_glm_val, "GLM Gaussian"))

# Instantiate a gamma family model with the default link function.
# Test set
glm_gamma = sm.GLM(train_y.values, train_X_min_max , family=sm.families.Gamma(
res_gamma = glm_gamma.fit()
predict_glm_gamma_train = res_gamma.predict()
MSE = mean_squared_error(train_y, predict_glm_gamma_train)
sMSE_gamma_train = np.sqrt(MSE)

# For validation set
predict_glm_gamma_val = res_gamma.predict(val_X_min_max)
MSE = mean_squared_error(val_y, predict_glm_gamma_val)
sMSE_gamma_val = np.sqrt(MSE)

display(analyze_results(val_y.values, predict_glm_gamma_val, "GLM Gamma"))
```



	Interval	Count	RMSE	Max_Difference
0	$0.0kto$100.0k	169	54761.1	157885.7
1	$100.0kto$200.0k	575	53979.2	289414.0
2	$200.0kto$300.0k	315	67431.4	239589.4
3	$300.0kto$400.0k	131	80433.1	302077.6
4	$400.0kto$500.0k	73	108900.2	419389.7
5	$500.0kto$600.0k	38	134325.1	369244.4
6	$600.0kto$700.0k	30	140095.5	531644.1
7	$700.0kto$800.0k	22	140717.5	348224.7
8	$800.0kto$900.0k	14	223111.5	400643.2
9	$900.0k to Maximum	25	352097.7	763736.8
10	All Data	1392	88221.7	763736.8

```
/Users/kdevoe/Documents/CS/Masters/AAI500/FInal_Project/aai-500-final-group-4/
env/lib/python3.10/site-packages/statsmodels/genmod/families/links.py:13: Futu
reWarning:

The identity link alias is deprecated. Use Identity instead. The identity link
alias will be removed after the 0.15.0 release.

/Users/kdevoe/Documents/CS/Masters/AAI500/FInal_Project/aai-500-final-group-4/
env/lib/python3.10/site-packages/statsmodels/genmod/generalized_linear_model.p
y:307: DomainWarning:

The identity link function does not respect the domain of the Gamma family.
```

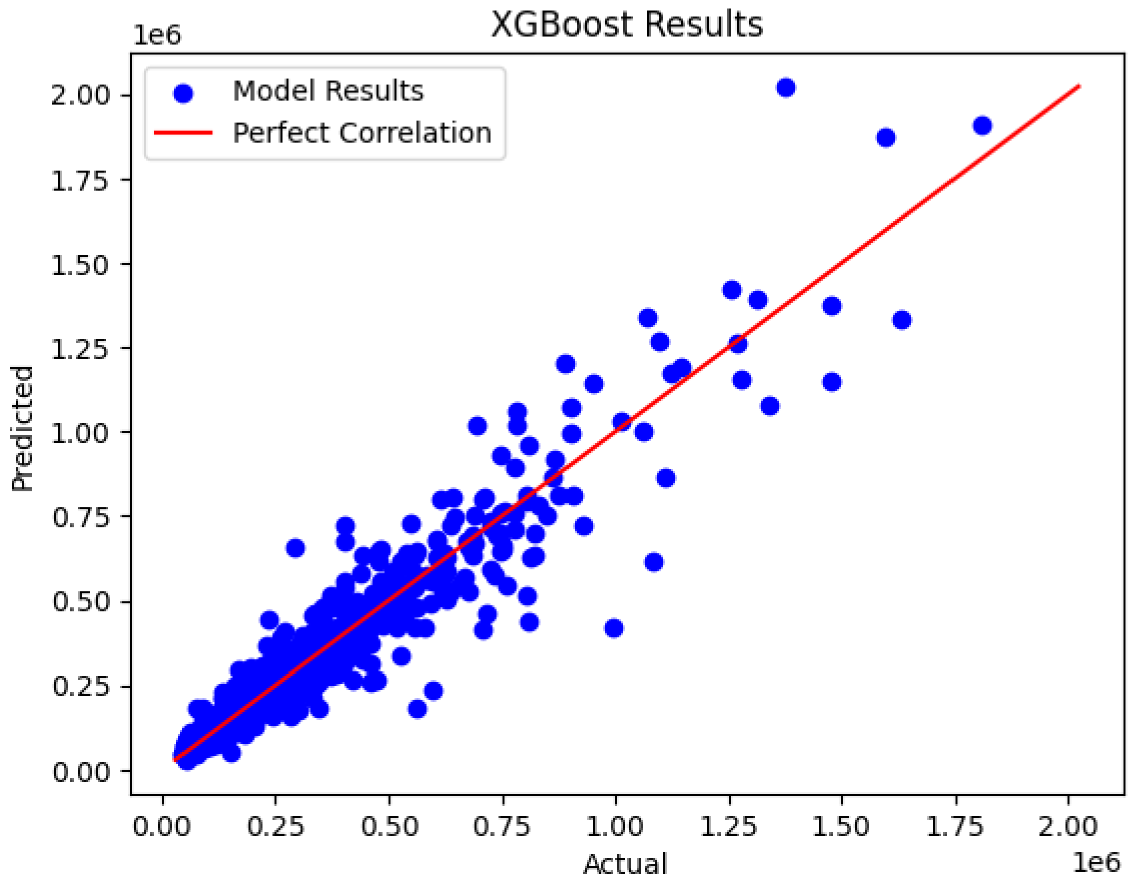


	Interval	Count	RMSE	Max_Difference
0	$0.0kto$100.0k	169	30192.7	109049.2
1	$100.0kto$200.0k	575	33874.8	143844.9
2	$200.0kto$300.0k	315	50028.7	242136.8
3	$300.0kto$400.0k	131	93307.8	406866.1
4	$400.0kto$500.0k	73	105710.6	349132.3
5	$500.0kto$600.0k	38	150158.6	385822.1
6	$600.0kto$700.0k	30	144080.8	302136.0
7	$700.0kto$800.0k	22	197070.5	352957.2
8	$800.0kto$900.0k	14	278715.3	477259.7
9	$900.0k to Maximum	25	493915.6	983295.3
10	All Data	1392	96931.8	983295.3

XGBoost Evaluation

In [ ]:

```
from xgboost import XGBRegressor

model = XGBRegressor()

# fit the model on the whole dataset
model.fit(train_X, train_y.values)

# make a single prediction
predict_xgboost_val = model.predict(val_X)

display(analyze_results(val_y.values, predict_xgboost_val, 'XGBoost'))
```



	Interval	Count	RMSE	Max_Difference
0	0.0kto100.0k	169	21462.3	106746.7
1	100.0kto200.0k	575	24365.9	127943.8
2	200.0kto300.0k	315	42257.2	365805.9
3	300.0kto400.0k	131	54280.0	160963.2
4	400.0kto500.0k	73	91172.6	321746.4
5	500.0kto600.0k	38	113522.7	380006.2
6	600.0kto700.0k	30	100812.1	327768.2
7	700.0kto800.0k	22	146740.0	295872.4
8	800.0kto900.0k	14	178449.7	370654.9
9	$900.0k to Maximum	25	259402.3	648382.9
10	All Data	1392	62115.9	648382.9

Sequential Neural Network Evaluation

In [ ]:
```
# Constants

BATCH_SIZE = 1
EPOCHS = 10
LR = 0.01
```

In [ ]:
```
# Model definition
class Model(nn.Module):
    def __init__(self, input_size, output_size=1):
        super().__init__()

        self.hidden_dim = output_size + (input_size - output_size) // 2

        self.fc_1 = nn.Linear(input_size, self.hidden_dim)
        self.fc_2 = nn.Linear(self.hidden_dim, output_size)

    def forward(self, x):

        out = self.fc_1(x)
        out = F.relu(out)
        out = self.fc_2(out)

        return out

neural_model = Model(train_X.shape[1])
print(neural_model)
```

```
Model(
  (fc_1): Linear(in_features=51, out_features=26, bias=True)
  (fc_2): Linear(in_features=26, out_features=1, bias=True)
)
```

Run on GPU if available

In [ ]:
```
is_cuda = torch.cuda.is_available()

if is_cuda:
    device = torch.device("cuda")
    print("GPU is available")

else:
    device = torch.device("cpu")
    print("GPU not available, CPU used")

neural_model.to(device)
```

```
GPU not available, CPU used
```

Out[ ]:
```
Model(
  (fc_1): Linear(in_features=51, out_features=26, bias=True)
  (fc_2): Linear(in_features=26, out_features=1, bias=True)
)
```

In [ ]:
```
# Define the loss function and the optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(neural_model.parameters(), lr=LR)
```

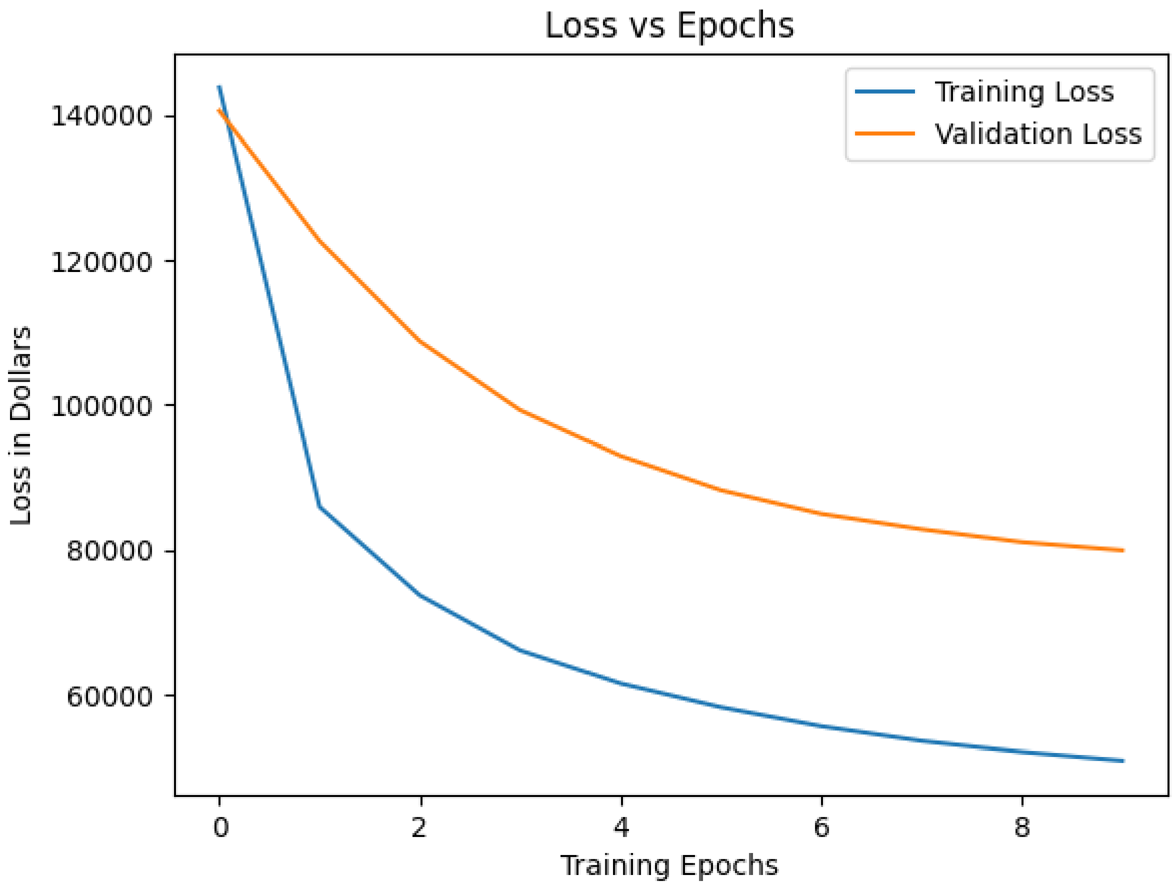
In [ ]:
```
# Training loop

def training_loop(model, train_X, train_y, val_X, val_y, num_epochs = 1, batch_

    num_batches = len(train_X) // batch_size
```

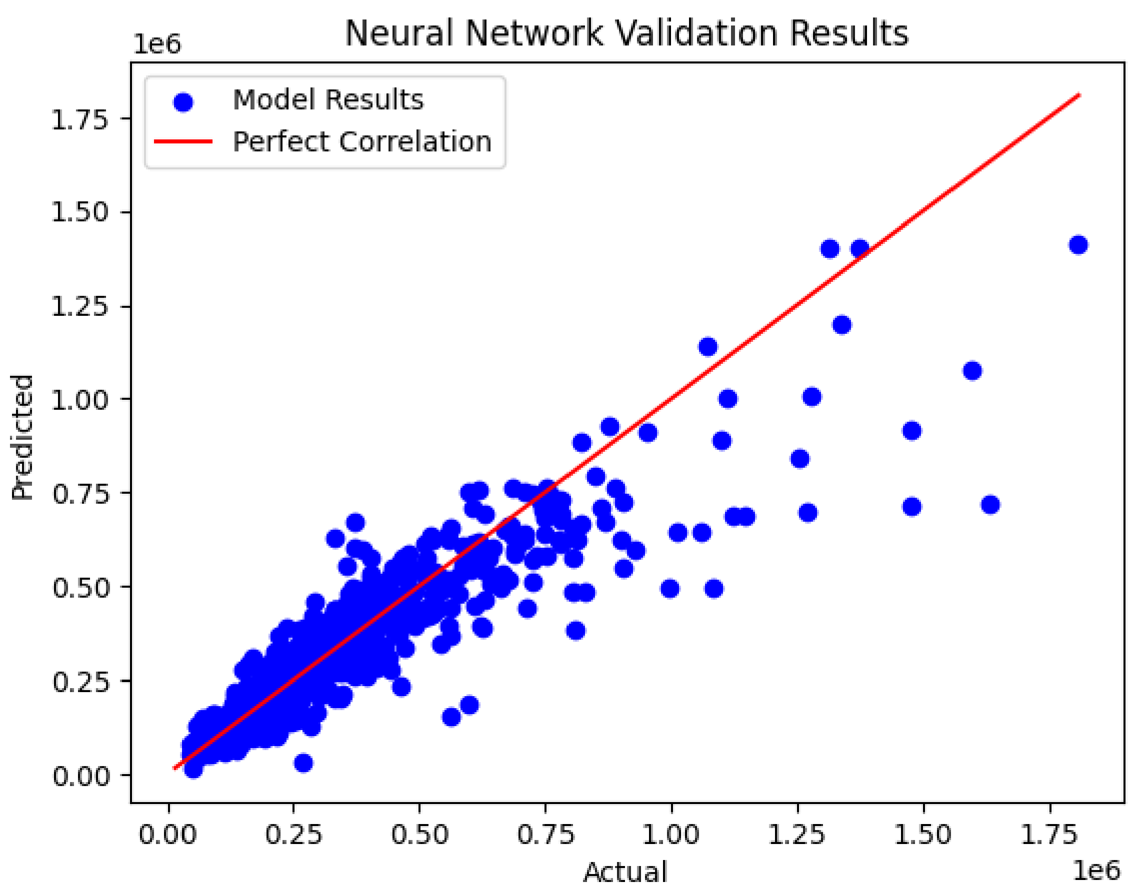
```
    train_losses = []
    val_losses = []
    total_loss = 0

    for epoch in tqdm(range(num_epochs)):  # loop over the dataset for each epo
        for batch in range(num_batches):
            # get the inputs; data is a list of [inputs, labels]
            start_index = batch * batch_size
            end_index = start_index + batch_size
            inputs = torch.from_numpy(train_X[start_index: end_index]).float().
            inputs.requires_grad = True
            labels = torch.from_numpy(np.asarray(train_y[start_index: end_index
            labels.requires_grad = True

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = model(inputs)
            loss = torch.sqrt(criterion(outputs, labels))
            loss.backward()
            optimizer.step()

            total_loss += np.mean(loss.item())

        # Get the validation losses
        val_outputs = model(torch.from_numpy(val_X).float().reshape(len(val_X),
        val_loss = torch.sqrt(criterion(val_outputs, torch.from_numpy(np.asarra

        average_loss = total_loss / num_batches
        train_losses.append(average_loss)
        val_losses.append(val_loss.detach().numpy())
        total_loss = 0  # Reset the total loss for the next epoch

    return train_losses, val_losses
```

In [ ]:

```
train_losses, val_losses = training_loop(neural_model, train_X, train_y, val_X,

# Get predictions on the validation set
val_outputs = neural_model(torch.from_numpy(val_X).float().reshape(len(val_X),
val_outputs = val_outputs.detach().numpy()

plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.title('Loss vs Epochs')
plt.xlabel('Training Epochs')
plt.ylabel('Loss in Dollars')
plt.legend()

plt.show()

# Convert numpy ndarray to list
val_outputs = val_outputs.tolist()
val_outputs = [item for sublist in val_outputs for item in sublist]

display(analyze_results(val_y.values, val_outputs, "Neural Network Validation")
```

```
100%|██████████| 10/10 [00:27<00:00,  2.78s/it]
```





	Interval	Count	RMSE	Max_Difference
0	$0.0kto$100.0k	169	24437.9	80186.8
1	$100.0kto$200.0k	575	28113.2	139668.3
2	$200.0kto$300.0k	315	49869.0	235489.3
3	$300.0kto$400.0k	131	74722.0	301257.3
4	$400.0kto$500.0k	73	76979.6	227074.2
5	$500.0kto$600.0k	38	128501.1	408861.0
6	$600.0kto$700.0k	30	107937.5	240224.3
7	$700.0kto$800.0k	22	117832.5	271034.2
8	$800.0kto$900.0k	14	215711.3	424131.9
9	$900.0k to Maximum	25	422006.1	912800.9
10	All Data	1392	79910.5	912800.9