Title: Strathcona House Value Predictor

Gilbert Akuja, Tianjiao Jiang, Thamer Aldawood & Yajing Liu

Summary

Our team will be working on predicting house prices using the 2023 Property Tax Assessment dataset from Strathcona County Open Data portalCounty (2023). The dataset provides a wealth of information about houses, including attributes like size, location, and other features. By leveraging this data, we aim to build a robust predictive model that accurately estimates house values.

Introduction

The team will be using `Ridge` which is a linear model to predict the value of houses. Ridge is a regularization model that is used for predictive modeling and mitigates over fitting, improves model stability especially when features are highly correlatedHoerl and Kennard (1970). Ridge helps create robust model that generalize well to new data. The question we aim to answer: Can we predict house prices using publicly available housing data , and which features most influence the predictions? Data description: For this project we are going to use the 2023 Property Tax Assessment from Strathcona County Open Data portalCounty (2023). The data set contains the following attributes related to the different houses. The variables we selected for the model are: `meters` - numeric variable that show the size of the house `garage` - categorical variable where Y means there is a garage and N means no garage. `firepl` - categorical variable where Y means there is a fireplace and N means no fireplace `bdevl` - categorical variable where Y meas the building was evaluated and N means it was not evaluated The data set was chosen for its rich feature set, adequate sample size, and public availability making it suitable for building a predictive model.

Methods & Results

We used Ridge Regression to predict house values based on features such as building size, garage presence, and building evaluation. Ridge regression, as outlined by Hoerl and Kennard (1970), is particularly useful in addressing multicollinearity. Model selection and evaluation

were facilitated by Scikit-Learn's robust tools Learn (2024). Exploratory visualizations were created using Altair "Altair Tutorial, Exploratory Data Visualization with Altair" (2024). The report is generated using tools Quart Team (2024).

1. Import all the necessary libraries for data analysis

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
#import altair_ally as aly
import altair as alt
import os
```

2. Read in dataset

The code reads a CSV file "named 2023 Property Tax Assessment" into a pandas DataFrame and filters it to include only the features we are going to evaluateCounty (2023). The resulting DataFrame contains these specific features for further analysis. Table 1 displays the first few rows of the dataset after filtering for relevant columns.

```
url = "https://hub.arcgis.com/api/v3/datasets/e3c5b04fccdc4ddd88059a8c0b6d8160_0/downloads/da
housing_df = pd.read_csv(url)

# Saves the dataframe to .csv in the data/ directory
housing_df.to_csv("../data/2023_Property_Tax_Assessment.csv")

# Takes only the columns we need
housing_df = housing_df[['meters','garage','firepl','bsmt','bdevl','assess_2022']]

housing_df
```

Table 1: Preview of the 2023 Property Tax Assessment dataset after selecting relevant columns.

	meters	garage	firepl	bsmt	bdevl	assess_2022
0	150.590	Y	Y	Y	N	382460
1	123.560	N	Y	N	N	280370
2	104.980	N	N	N	N	402000
3	66.611	N	N	N	N	3690
4	123.830	Y	Y	Y	Y	295910

Table 1: Preview of the 2023 Property Tax Assessment dataset after selecting relevant columns.

	meters	garage	firepl	bsmt	bdevl	assess_2022
...	...	...	...	...	...	...
35756	121.330	Y	Y	Y	Y	363000
35757	132.470	Y	Y	Y	N	355000
35758	121.330	Y	Y	Y	N	347000
35759	121.330	Y	Y	Y	Y	363000
35760	120.860	Y	Y	Y	Y	344000

```
# Validation for correct data file format
file_path = "data/2023_Property_Tax_Assessment.csv"

if not file_path.endswith(".csv"):
    raise ValueError("File format not supported.")
else:
    print("  File format validation passed: File is a CSV.")

# Validation for correct column names
expected_cols = {"meters", "garage", "firepl", "bsmt", "bdevl", "assess_2022"}
actual_cols = set(housing_df.columns)

if actual_cols == expected_cols:
    print("  Column name validation passed: All expected columns are present.")
else:
    missing = expected_cols - actual_cols
    extra = actual_cols - expected_cols
    print("Column names validation failed:")
    if missing:
        print(f" Missing columns: {missing}")
    if extra:
        print(f" Extra columns: {extra}")

# Validation for No Empty Observations
empty = housing_df.isna().all(axis = 1).sum()

if empty == 0:
    print("  Empty observations validation passed: No empty rows.")
else:
    print(f" Empty observations validation failed: Found {empty} empty rows.")

#Validation for missingness not beyond expected threshold
```

```
expected_threshold = 0
missing = housing_df.isna().mean()

all_cols_passed = True

for col, percentage in missing.items():
    if percentage > expected_threshold:
        all_cols_passed = False
        print(f"Column '{col}' exceeds the missingness expected threshold ({percentage:.2%} m
if all_cols_passed:
    print(f" Missingness validation passed for all columns.")

#Validation for correct data types in each column

expected_dtypes = { "meters": "float64","garage": "object","firepl": "object","bsmt": "object

dtype_passed = True

for col, dtype in expected_dtypes.items():
    if housing_df[col].dtypes != dtype:
        dtype_passed = False
        print(f"Data type validation failed: Column '{col}' is of type {housing_df[col].dtype
if dtype_passed:
    print(" Data type validation passed for all columns.")

# Validation for no duplicate observations
duplicates = housing_df.duplicated().sum()

if duplicates == 0:
    print(" Duplicate observation validation passed: No duplicate rows found.")
else:
    print(f"Duplicate observation validation failed: Found {duplicates} duplicate rows.")
    # Dropping duplicates
    housing_df = housing_df.drop_duplicates()
    print(" Duplicates have been removed from the DataFrame.")

# Validation for no outliers or anomalous values
outliers_detected = False
outlier_threshold = 5000

# Identify and drop outliers if the threshold is exceeded
for col in housing_df.select_dtypes(include=["float64", "int64"]).columns:
```

```
    q1 = housing_df[col].quantile(0.25)
    q3 = housing_df[col].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    outliers = housing_df[(housing_df[col] < lower_bound) | (housing_df[col] > upper_bound)]
    num_outliers = len(outliers)

    if num_outliers > outlier_threshold:
        outliers_detected = True
        print(f"Outlier validation failed: Column '{col}' has {num_outliers} outliers (thresh
        # Dropping the outliers
        housing_df = housing_df[~((housing_df[col] < lower_bound) | (housing_df[col] > upper_
        print(f"  Outliers in column '{col}' have been removed from the DataFrame.")
    elif num_outliers > 0:
        print(f"Warning: Column '{col}' has {num_outliers} outliers, within acceptable thresh

if not outliers_detected:
    print("  Outlier validation passed: No columns exceed the outlier threshold.")
else:
    print("  Outliers exceeding the threshold have been removed.")
```

```
  File format validation passed: File is a CSV.
  Column name validation passed: All expected columns are present.
  Empty observations validation passed: No empty rows.
  Missingness validation passed for all columns.
  Data type validation passed for all columns.
Duplicate observation validation failed: Found 2425 duplicate rows.
  Duplicates have been removed from the DataFrame.
Warning: Column 'meters' has 885 outliers, within acceptable threshold (5000).
Warning: Column 'assess_2022' has 2048 outliers, within acceptable threshold (5000).
  Outlier validation passed: No columns exceed the outlier threshold.
```

3. Visualization for categorical features

The code enables VegaFusion to optimize Altair data transformations for visualizations"Altair Tutorial, Exploratory Data Visualization with Altair" (2024). It then creates a distribution plot of categorical features in the `housing_df` DataFrame. As shown in Figure 1, the distribution of categorical features provides insights into their overall counts.

```
alt.data_transformers.enable("vegafusion")

grg0 = alt.Chart(housing_df).mark_bar().encode(
    x = alt.X('garage', title='Garage'),
    y = alt.Y('count()', scale = alt.Scale(domain=[0,35000]), title='House value'),
)


frp0 = alt.Chart(housing_df).mark_bar().encode(
    x = alt.X('firepl', title='Fireplace'),
    y = alt.Y('count()', scale = alt.Scale(domain=[0,35000]), title='House value'),
)

bst0 = alt.Chart(housing_df).mark_bar().encode(
    x = alt.X('bsmt', title='Basement'),
    y = alt.Y('count()', scale = alt.Scale(domain=[0,35000]), title='House value'),
)

bdl0 = alt.Chart(housing_df).mark_bar().encode(
    x = alt.X('bdevl', title='Building evaluation'),
    y = alt.Y('count()', scale = alt.Scale(domain=[0,35000]), title='House value'),
)

(grg0 | frp0 | bst0 | bdl0).properties(
    title="Counts of categorical features"
)
```

```
alt.HConcatChart(...)
```

Figure 1: Distribution of categorical features (Garage, Fireplace, Basement, and Building Evaluation).

The code generates scatter plots to visualize the relationship between house assessment values (`assess_2022`) and four categorical features: `garage`, `firepl`, `bsmt`, and `bdevl`. As depicted in Figure 2, the scatter plots illustrate the relationship between house value assessments and categorical features.

```
grg = alt.Chart(housing_df).mark_point().encode(
    x = alt.X('garage', title='Garage'),
    y = alt.Y('assess_2022', title='House value'),
)
```

```
frp = alt.Chart(housing_df).mark_point().encode(
    x = alt.X('firepl', title='Fireplace'),
    y = alt.Y('assess_2022', title='House value'),
)

bst = alt.Chart(housing_df).mark_point().encode(
    x = alt.X('bsmt', title='Basement'),
    y = alt.Y('assess_2022', title='House value'),
)

bdl = alt.Chart(housing_df).mark_point().encode(
    x = alt.X('bdevl', title='Building evaluation'),
    y = alt.Y('assess_2022', title='House value'),
)

(grg | frp | bst | bdl).properties(
    title="House value assessment per categorical feature")
```

```
alt.HConcatChart(...)
```

Figure 2: Scatter plots showing house value assessments for categorical features: Garage, Fireplace, Basement, and Building Evaluation.

4. Prepare data for training and create column transformer for feature transformation

Visualizing the distribution of the target variable, assess_2022, using a histogram and Kernel Density Estimate (KDE) plot is a crucial step in data exploration. It helps us understand the overall distribution, identify patterns such as skewness, kurtosis, or multiple peaks, and detect outliers. By comparing the histogram and KDE, we can assess if the distribution meets the assumptions of our chosen modeling techniques. This visualization also guides decisions about preprocessing, such as applying transformations to address skewness or choosing the right algorithm for modeling. Additionally, it can uncover data quality issues, like sparse regions or missing values, prompting necessary corrections. As depicted in Figure 3, the combined visualizations provide valuable insights into the distribution and relationship between property sizes (meters) and house assessment values (assess_2022). The scatter plot showcases how larger properties tend to have higher assessment values, while the histogram and KDE plot reveal the overall distribution and density of assessment values in the dataset.

```
scatter = alt.Chart(housing_df).mark_point().encode(
    y = alt.Y('meters', title="Property size (meters)"),
```

```
    x = alt.X('assess_2022', title="Assessment Value"),
    color = alt.Color('meters', title="Property size (meters)").scale(scheme='viridis')
).properties(
    title="Scatter plot of Property Size and Assessment Values (2022)"
)

# Create a histogram and KDE plot for the 'assess_2022' column
histogram = alt.Chart(housing_df).mark_bar().encode(
    alt.X("assess_2022:Q", bin=alt.Bin(maxbins=2000), title="Assessment Value").scale(domain=
    alt.Y("count():Q", title="Frequency"),
).properties(
    title="Distribution of House Assessment Values (2022)"
)

kde = alt.Chart(housing_df).transform_density(
    "assess_2022",
    as_=["assess_2022", "density"],
    counts=True
).mark_line(color="red").encode(
    alt.X("assess_2022:Q", title="Assessment Value").scale(domain=(0, 2000000), clamp=True),
    alt.Y("density:Q", title="Density")
).properties(
    title="Line plot of KDE House Assessment Values (2022)"
)

# Combine the histogram and KDE plot
combined_chart = histogram | kde | scatter
combined_chart.show()
```

```
alt.HConcatChart(...)
```

Figure 3: Distribution and relationships of house assessment values (2022), showing scatter plot, histogram, and kernel density estimation.

The scatter plot shows that the vast majority of our data points are concentrated within the $0 <$ meters < 2000 range, with some more within the $2000 <$ meters < 5000 range, and there are a few outliers in the meters > 5000 range that make it difficult to get a closer look on the majority of our data. We can see a similar story on the assess_2022 feature as it has the vast majority of points in the low-mid range and some outlying extreme values.

The code assigns the housing_df DataFrame into training and test datasets using an 70-30 split. The code then categorizes features into categorical features (e.g., `garage`, `firepl`,

`bsmt`, `bdevl`) and numeric features (e.g., `meters`), applies one-hot encoding transforamtion for categorical features and standardScalar transformation for numeric features by using Scikit-learnLearn (2024). The code created a column transformer `preprocessor` through combining these transforamtions to apply to the dataset and visualizes the `preprocessor`.

We performed analysis to identify any potential issues with the relationships between features and the target, as well as between the features themselves. By detecting anomalous correlations, we can uncover redundant or highly correlated features that might lead to multicollinearity, which can negatively affect the performance of machine learning models. In particular, removing or adjusting features that are too strongly correlated ensures that the model can learn meaningful, independent relationships, improving its generalization ability and interpretability. Additionally, examining correlations helps us assess if certain features are disproportionately influencing the target variable, which could indicate bias or overfitting.

```
train_df, test_df = train_test_split(housing_df, test_size=0.3, random_state=123)
```

```
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Lists of feature names
categorical_features = ['garage', 'firepl', 'bsmt', 'bdevl']
numeric_features = ['meters']
# Create the column transformer
preprocessor = make_column_transformer(
    (OneHotEncoder(), categorical_features),  # One-hot encode categorical columns
    (StandardScaler(), numeric_features),  # Standardize numeric columns
)

# Show the preprocessor
preprocessor
```

```
ColumnTransformer(transformers=[('onehotencoder', OneHotEncoder(),
                                 ['garage', 'firepl', 'bsmt', 'bdevl']),
                                ('standardscaler', StandardScaler(),
                                 ['meters'])])
```

Validation step

```
# Define the categorical features and their expected values
categorical_features = ['garage', 'firepl', 'bsmt', 'bdevl']
```

```
expected_values = {'Y', 'N'}

# Initialize a flag to track validation status
all_categories_valid = True

# Validate each categorical feature in the train_df
for feature in categorical_features:
    unique_values = set(train_df[feature].dropna().unique())  # Get unique values excluding N
    unexpected_values = unique_values - expected_values  # Find unexpected values

    if unexpected_values:
        print(f"  Unexpected values in column '{feature}' in train_df: {unexpected_values}")
        all_categories_valid = False
    else:
        print(f"  Category levels validation passed for column '{feature}' in train_df.")

# Final message
if all_categories_valid:
    print("  All categorical columns in train_df have the expected levels ('Y' and 'N').")
else:
    print("  Some categorical columns in train_df have unexpected values. Please review.")
```

```
  Category levels validation passed for column 'garage' in train_df.
  Category levels validation passed for column 'firepl' in train_df.
  Category levels validation passed for column 'bsmt' in train_df.
  Category levels validation passed for column 'bdevl' in train_df.
  All categorical columns in train_df have the expected levels ('Y' and 'N').
```

Check for anomalous correlations between target and features

```
# Apply preprocessing to train_df
transformed_train = preprocessor.fit_transform(train_df)

# Add the target variable back to the DataFrame
transformed_train_df = pd.DataFrame(transformed_train)
transformed_train_df.columns = ['Garage_Y', 'Garage_N', 'Fireplace_Y', 'Fireplace_N','Basemen
                                'BuildingEva_Y', 'BuildingEva_N','meters']
# Add the target column back (assess_2022) to the transformed DataFrame
transformed_train_df['assess_2022'] = train_df['assess_2022']

# Compute correlation with the target variable
```

```
correlation_with_target = transformed_train_df.corr()["assess_2022"].drop("assess_2022")

# Identify features with anomalous correlations (correlation > 0.9 or < -0.9)
threshold = 0.9
anomalous_features = correlation_with_target[
    (correlation_with_target.abs() > threshold)
]

if anomalous_features.empty:
    print("  No anomalous correlations found between target and features.")
else:
    print("  Anomalous correlations detected:")
    print(anomalous_features)
```

```
  No anomalous correlations found between target and features.
```

Check for anomalous correlations between features

```
correlation_matrix = transformed_train_df.corr()
threshold = 0.9

# Define pairs of features that are binary inverses of each other (which are expected to be h
exclude_pairs = [
    ('Garage_Y', 'Garage_N'),
    ('Fireplace_Y', 'Fireplace_N'),
    ('Basement_Y', 'Basement_N'),
    ('BuildingEva_Y', 'BuildingEva_N')
]

# Flag to check if any anomalous correlations were found
anomalous_correlations_found = False

# Iterate over the correlation matrix to check for high correlations
for col1 in correlation_matrix.columns:
    for col2 in correlation_matrix.columns:
        if col1 != col2:
            # Skip pairs that are binary inverses
            if (col1, col2) in exclude_pairs or (col2, col1) in exclude_pairs:
                continue
            # Check if correlation is above threshold
            if abs(correlation_matrix[col1][col2]) > threshold:
```

```
                print(f"Warning: High correlation detected between {col1} and {col2}.")
                anomalous_correlations_found = True

# If no anomalous correlations are found, print the confirmation message
if not anomalous_correlations_found:
    print("  No anomalous correlations found between target and features.")
```

```
  No anomalous correlations found between target and features.
```

Note: When performing correlation analysis on the dataset, it is important to exclude pairs of binary features that are inherently correlated due to their inverse relationship. For example, in the case of features like Garage_Y and Garage_N, one feature will always be 1 when the other is 0, and vice versa. These binary features are designed to represent the same information in different forms, and their high correlation is expected. Thus, it is not an anomaly and should not be flagged as such. To address this, we exclude these pairs from the correlation analysis, ensuring that only truly anomalous correlations—those that do not follow expected patterns—are identified. By doing so, we can avoid mistakenly flagging normal relationships between inverse binary features while still identifying potential issues in the remaining feature correlations.

5. Train, cross validate and evaluate a Ridge regression model

The code splits the features and target variable (`assess_2022`) into training and testing datasets.

```
X_train = train_df.drop(columns=["assess_2022"])
X_test = test_df.drop(columns=["assess_2022"])
y_train = train_df["assess_2022"]
y_test = test_df["assess_2022"]
```

The code creates a pipeline combining the column transformer (`preprocessor`) and the Ridge Regression modelHoerl and Kennard (1970). Using 5-fold cross-validation on the training data, the code evaluates the pipeline on multiple metrics and computes train and validation scores. Finally, it outputs the aggregated train and validation scores to assess the model's performanceLearn (2024). Table 2 summarizes the cross-validation results, including the mean and standard deviation of train and validation scores across 5 folds.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import (
    cross_val_score,
    cross_validate
)
from sklearn.pipeline import make_pipeline

# The Ridge model pipeline
pipeline = make_pipeline(preprocessor, Ridge())

# The mean and std of the cross validated scores for all metrics as a dataframe
cross_val_results = pd.DataFrame(cross_validate(pipeline, X_train, y_train, cv=5, return_tra

# Show the train and validation scores
cross_val_results
```

Table 2: Cross-validation results showing the mean and standard deviation for train and validation scores across 5 folds.

	mean	std
fit_time	0.010	0.000
score_time	0.003	0.000
test_score	0.564	0.235
train_score	0.575	0.078

The code fits the pipeline on the training dataset (`X_train` and `y_train`) to train the Ridge Regression modelHoerl and Kennard (1970). Then evaluates the trained pipeline on the test dataset (`X_test` and `y_test`) which calculates the R^2 (coefficient of determination) to measure how well the model explains the variance in the test data.

```
pipeline.fit(X_train, y_train)
pipeline.score(X_test, y_test)
```

```
0.536298068928245
```

6. Predict housing prices with new data

This code creates a Pandas DataFrame containing information about 10 houses that we wish to predict the value of. Table 3 summarizes the attributes of ten houses, including property size and various categorical features, used for predictions.

```
ten_houses = {
    'meters' : [174.23, 132.76, 90.82, 68.54, 221.30, 145.03, 102.96, 164.28, 142.79, 115.94
    'garage' : ['Y', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'Y', 'N', 'Y'],
    'firepl' : ['Y', 'N', 'N', 'N', 'Y', 'N', 'N', 'Y', 'Y', 'N'],
    'bsmt' : ['Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y', ],
    'bdevl' : ['N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', ]
    }

X_predict = pd.DataFrame(ten_houses)
X_predict
```

Table 3: Attributes of ten houses used for prediction.

	meters	garage	firepl	bsmt	bdevl
0	174.23	Y	Y	Y	N
1	132.76	Y	N	Y	Y
2	90.82	Y	N	N	Y
3	68.54	N	N	N	N
4	221.30	Y	Y	Y	Y
5	145.03	N	N	N	Y
6	102.96	N	N	Y	Y
7	164.28	Y	Y	N	N
8	142.79	N	Y	Y	N
9	115.94	Y	N	Y	Y

This code applies the trained pipeline to predict housing prices values based on the features in the dataFrame containing new data Hastie, Tibshirani, and Friedman (2009). The predictions are stored in a new pandas DataFrame.

```
y_predict = pipeline.predict(X_predict)
y_predict = pd.DataFrame(y_predict)
y_predict.columns = ['Predicted_Values']
y_predict = round(y_predict, 2)
```

The code combines the original features from the new data and the predicted values into a new pandas DataFrame. The new pandas DataFrame provides an overview of the predictions. Table 4 displays the predicted house values for the ten houses based on their attributes, including property size and categorical features.

```
predictions_df = pd.concat([X_predict,y_predict], axis = 1)

# Saves prediction to a csv file
predictions_df.to_csv("../data/ten_houses_predictions.csv")

predictions_df
```

Table 4: Predicted house values for the ten houses based on their attributes.

	meters	garage	firepl	bsmt	bdevl	Predicted_Values
0	174.23	Y	Y	Y	N	536986.06
1	132.76	Y	N	Y	Y	457896.92
2	90.82	Y	N	N	Y	361058.34
3	68.54	N	N	N	N	212406.79
4	221.30	Y	Y	Y	Y	700449.74
5	145.03	N	N	N	Y	445227.44
6	102.96	N	N	Y	Y	344014.38
7	164.28	Y	Y	N	N	515563.17
8	142.79	N	Y	Y	N	419237.26
9	115.94	Y	N	Y	Y	418244.15

7. Visualization for predictions

This code creates line charts to visualize the relationship between property size (`meters`) and predicted housing prices (`Predicted_Values`), colored by different categorical features. This visualization highlights how categorical features interact with property size to influence predicted house values“Altair Tutorial, Exploratory Data Visualization with Altair” (2024). As shown in Figure 4, property size has a significant impact on house value predictions, with notable variations across categorical features.

```
mtrs = alt.Chart(predictions_df).mark_line().encode(
    x = alt.X('meters', title="Property size"),
    y = alt.Y('Predicted_Values', title='Predicted Values'),
).properties(
    height = 200,
    width = 200,
    title = "Property size (m^2) vs value"
)

grg2 = alt.Chart(predictions_df).mark_line().encode(
```

```
    x = alt.X('meters', title="Property size"),
    y = alt.Y('Predicted_Values', title='Predicted Values'),
    color = alt.Color('garage', title="Garage")
).properties(
    height = 200,
    width = 200,
    title = "Garage"
)

frp2 = alt.Chart(predictions_df).mark_line().encode(
    x = alt.X('meters', title="Property size"),
    y = alt.Y('Predicted_Values', title='Predicted Values'),
    color = alt.Color('firepl', title="Fireplace")
).properties(
    height = 200,
    width = 200,
    title = "Fireplace"
)

bst2 = alt.Chart(predictions_df).mark_line().encode(
    x = alt.X('meters', title="Property size"),
    y = alt.Y('Predicted_Values', title='Predicted Values'),
    color = alt.Color('bsmt', title="Basement")
).properties(
    height = 200,
    width = 200,
    title = "Basement"
)

bdl2 = alt.Chart(predictions_df).mark_line().encode(
    x = alt.X('meters', title="Property size"),
    y = alt.Y('Predicted_Values', title='Predicted Values'),
    color = alt.Color('bdevl', title="Building evaluation")
).properties(
    height = 200,
    width = 200,
    title = "Building evaluation"
)

(mtrs & (grg2 | frp2) & (bst2 | bdl2)).properties(
    title = "Correlations between property size and house value, colored by different charact
)
```

```
alt.VConcatChart(...)
```

Figure 4: Line charts showing correlations between property size and house value predictions, colored by various categorical features.

Discussion

Our findings suggest that the aforementioned 10 houses are expected to be valued at:

```
y_predict
```

	Predicted_Values
0	536986.06
1	457896.92
2	361058.34
3	212406.79
4	700449.74
5	445227.44
6	344014.38
7	515563.17
8	419237.26
9	418244.15

We have also noticed that there is a correlation between a house's price and its property's size, whether or not it has a garage, fireplace, basement, and whether or not the building has been evaluated. This is consistent with our expectations as the larger a property is, and the more features it has (basement, garage, etc.) the higher its value becomesFastExpert (2024). This also raises further questions such as what other features of a house or property affect its value? characteristics for future consideration include: Number of bedrooms, number of bathrooms, indoor space, outdoor space, and number of floors.

References

"Altair Tutorial, Exploratory Data Visualization with Altair." 2024. 2024. https://altair-viz.github.io/altair-tutorial/README.html.

County, Strathcona. 2023. "2023 Property Tax Assessment." https://www.strathcona.ca/services/assessment/.

FastExpert. 2024. "How Much Value Does a Garage Add to a House?" 2024. https://www.fastexpert.com/blog/how-much-value-does-a-garage-add-to-a-house/.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

Hoerl, Arthur E., and Robert W. Kennard. 1970. "Ridge Regression: Biased Estimation for Nonorthogonal Problems." *Technometrics* 12 (1): 55–67.

Learn, Scikit. 2024. "Model Selection and Evaluation." 2024. https://scikit-learn.org/stable/model_selection.html.

Team, Quarto Development. 2024. "Quarto: An Open-Source Scientific and Technical Publishing System." https://quarto.org/.