UNIVERSITY OF MARYLAND

ROBERT H. SMITH SCHOOL OF BUSINESS

Project Report
Next Home: House Price Prediction
Spring 2024

Code example to begin with:

```
import tensorflow_decision_forests as tfdf
import pandas as pd

dataset = pd.read_csv("project/dataset.csv")
tf_dataset = tfdf.keras.pd_dataframe_to_tf_dataset(dataset, l
abel="my_label")

model = tfdf.keras.RandomForestModel()
model.fit(tf_dataset)

print(model.summary())
```

Introduction:

In the realm of real estate, understanding housing market trends and factors influencing property prices is crucial for buyers, sellers, and investors alike.

Through meticulous data collection, processing, and analysis, Group 18 aimed to provide a comprehensive overview of housing sales trends, factors affecting property prices, and the overall condition of houses in the dataset. By leveraging advanced analytical techniques and visualization tools, we have tried to unravel the complexities of the housing market and present our findings in a clear and concise manner.

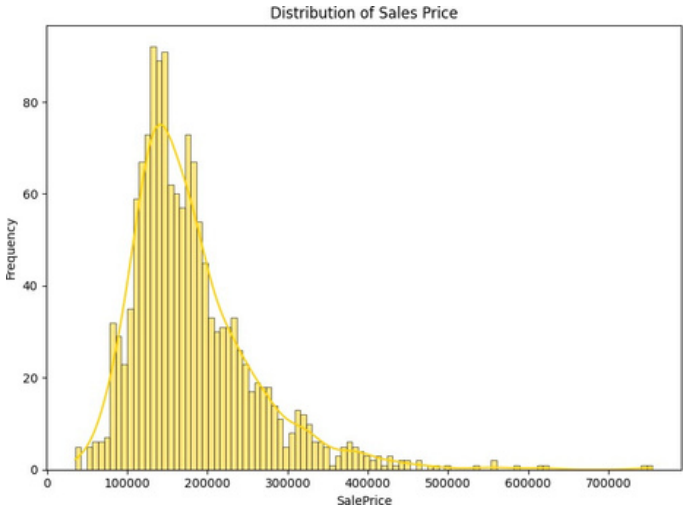
Code and Methods Used:

1. Exploratory Data Analysis - Visualizations

Our team began by conducting an in-depth examination of the dataset to understand its structure, variables, and potential insights it could offer. We utilized various statistical techniques and visualization tools to uncover patterns, trends, and relationships within the data.

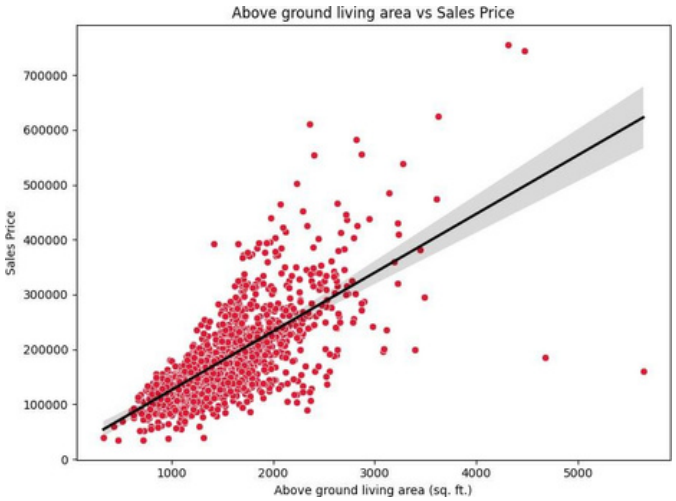
One key aspect of our exploratory data analysis was the creation of visualizations such as scatter plots, histograms, and box plots to visually represent the distribution of data points and relationships between variables. These visualizations allowed us to identify correlations, outliers, and trends that informed our subsequent analysis.

a) Histogram of Sales Price



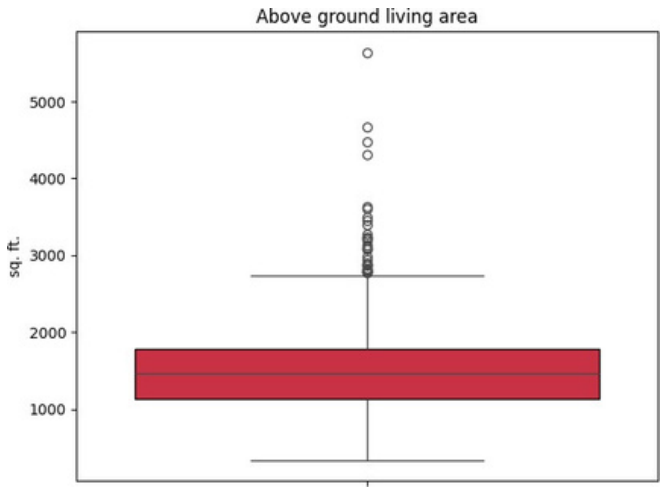
- The x-axis represents the sale price, likely ranging from around $0 to $700,000.
- The y-axis represents the frequency or count, indicating how many houses sold within each price range (bin) of the histogram.
- The histogram has a longer tail on the right side and the distribution is skewed to the right, meaning there are more houses sold on the lower end of the price range.
- There's a concentration of houses sold in the lower and middle price ranges (between $0 and $400,000). Fewer houses sold at the higher end of the range.

b) Scatter Plot of Sales Price V/s Above ground living area



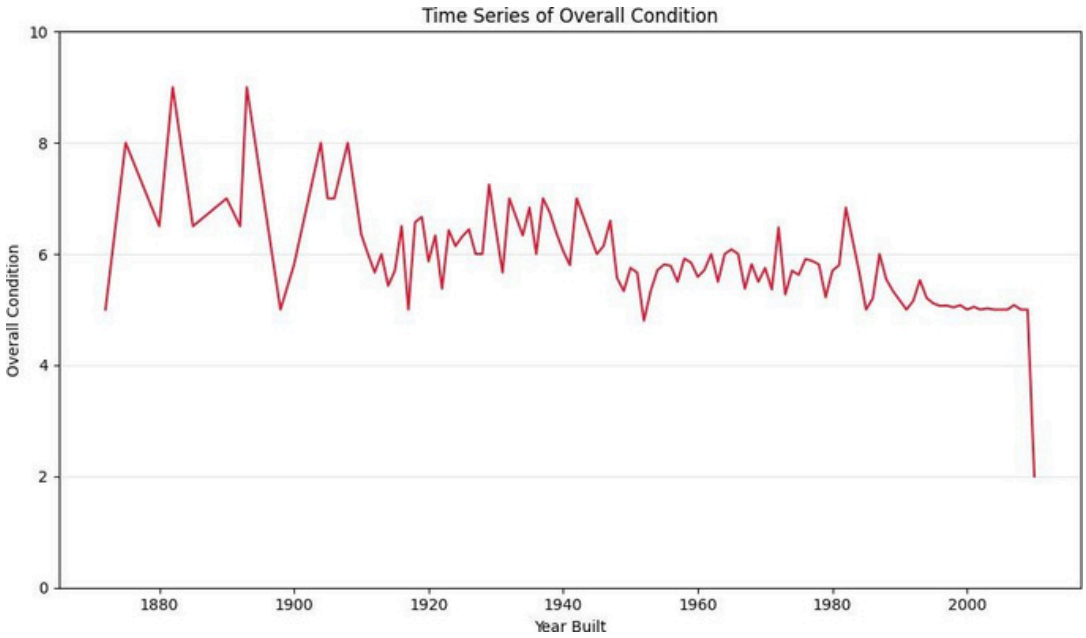
- Each dot in the scatter plot represents an individual house. The horizontal position of the dot indicates the house's above ground living area, and the vertical position indicates its sales price.
- There appears to be a positive correlation between the two variables. This means that as the above ground living area increases, the sales price also tends to increase. The data points show a general upward trend.
- The data points are not all clustered along a perfect straight line. There's a spread of data points around the upward trend. This indicates that while there's a positive relationship, other factors besides above ground living area also influence sales price.

c) Box plot of Above ground living area



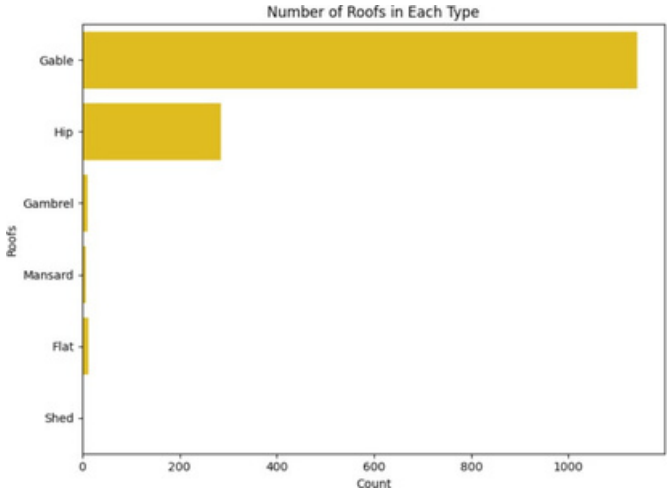
- This boxplot analyzes the distribution of the Above Ground living area of the houses.
- We can see that most houses are clustered between 100 and 2000. Although there are outliers at very high heights which is quite intriguing.

d) Condition of houses throughout the years



- The x-axis represents the year the house was built. The time range spans from 1870 to 2010.
- The y-axis represents the overall condition of the house. It is an index ranging from 0 to 10, 0 being the lowest and 10 being the highest.

e) Count of types of roof



- Gable seems to be the most preferred type of roof, with a count of more than 1000.
- Hip is next, with a count of about 250.
- Gambrel, Mansard and Flat have almost negligible count numbers.

Data Pre-Processing:

To ensure the integrity of our dataset and facilitate further analysis, we implemented a strategy for handling missing values in both categorical and numerical columns.

Handling Missing Values:

Categorical Columns: Missing values in categorical columns were imputed with 'Unknown' to maintain data integrity.

Numerical Columns: Missing values in numerical columns were imputed with the mean value to preserve the dataset's statistical properties.

```
# Imputing missing values in categorical columns with 'Unknown' for
column in categorical_columns_with_missing:
    train_data[column].fillna('Unknown', inplace=True)

# Imputing missing values in numerical columns with mean
numerical_imputer = SimpleImputer(strategy='mean')
train_data[numerical_columns_with_missing] =
numerical_imputer.fit_transform(train_data[numerical_columns_with_missing])
```

Model Training

We defined a function split_dataset to divide the dataset into training and testing data based on a specified test ratio. By randomly selecting test indices and computing training indices, the function efficiently splits the dataset. The resulting subsets, data_train and data_valid, enable effective model training and validation.

```
# Splitting the Data in Training and Validation Data def
split_dataset(dataset,          test_ratio=0.30):
        num_test = int(test_ratio * len(dataset))
        test_indices = np.random.choice(len(dataset), num_test, replace=False)
        train_indices = np.setdiff1d(np.arange(len(dataset)), test_indices) return
        dataset.iloc[train_indices],              dataset.iloc[test_indices]

data_train, data_valid = split_dataset(train_data)
print("{} examples in training, {} examples in testing.".format(len(data_train),
len(data_valid)))
```

1. Converting pandas DataFrames into TensorFlow Decision Forests (TF-DF) datasets tailored for regression tasks:

The code snippet showcases the transformation of a Pandas DataFrame into TensorFlow datasets tailored for regression tasks using the TensorFlow Decision Forests (TF-DF) library. By importing the necessary module tensorflow_decision_forests as tfdf, the code efficiently converts the data_train and data_valid Pandas DataFrames into TensorFlow datasets. The pd_dataframe_to_tf_datasetfunction from the tfdf.keras module is employed for this conversion process. Specifically, the label='SalePrice' parameter identifies the target variable within the dataset, crucial for regression analysis, while specifying task=tfdf.keras.Task.REGRESSION indicates that the objective is regression modeling. This conversion step is essential for preparing the data for subsequent model training and validation using TensorFlow Decision Forests, enabling the application of advanced regression techniques to analyze and predict housing sales trends and property prices effectively.

```
# Converting the dataset into a TensorFlow dataset import
tensorflow_decision_forests as tfdf
```

```
train_df = tfdf.keras.pd_dataframe_to_tf_dataset(data_train, label='SalePrice', task =
tfdf.keras.Task.REGRESSION)
valid_df = tfdf.keras.pd_dataframe_to_tf_dataset(data_valid, label='SalePrice', task =
tfdf.keras.Task.REGRESSION)
```

This code prepares the training and validation data for use with a TensorFlow Decision Forest regression model by converting them into TensorFlow datasets,this enables efficient training and potentially improves the overall workflow within the TensorFlow ecosystem.

```
# Converting the dataset into a TensorFlow dataset import
tensorflow_decision_forests as tfdf

train_df = tfdf.keras.pd_dataframe_to_tf_dataset(data_train, label='SalePrice', task =
tfdf.keras.Task.REGRESSION)
valid_df = tfdf.keras.pd_dataframe_to_tf_dataset(data_valid, label='SalePrice', task =
tfdf.keras.Task.REGRESSION)
```

2. Calculating RMSE and Model evaluation

This code defines a function to calculate Root Mean Squared Error (RMSE) and then uses it to evaluate the performance of two different models from the TensorFlow Decision Forests library on the validation data.

> The provided code snippet iterates through a list of models, fits each model to the training dataset (train_df), predicts the target variable for the validation dataset (valid_df), calculates the Root Mean Squared Error (RMSE) between the predicted and true values of the 'SalePrice' column in the validation data, and stores the RMSE results in a dictionary named rmse_results. This process allows for the evaluation of model performance based on RMSE.

```
from sklearn.metrics import mean_squared_error #

Creating function to calculate RMSE
def calculate_rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))
```

```
rmse_results = {}

# Choosing models from TensorFlow Decision Forest Package
models = [
        tfdf.keras.RandomForestModel(task=tfdf.keras.Task.REGRESSION),
        tfdf.keras.GradientBoostedTreesModel(task=tfdf.keras.Task.REGRESSION)
]


# Fitting models and calculating RMSE
for model in models:
        model.fit(train_df)
        y_pred = model.predict(valid_df)
        y_true = data_valid['SalePrice'].values
        rmse = calculate_rmse(y_true, y_pred)
        rmse_results[type(model). name ]_= rmse_
```

This part selects the best model with the lowest RMSE from the RMSE results stored in the rmse_results dictionary. It identifies the model with the minimum RMSE value and retrieves the corresponding RMSE score. Subsequently, the code displays the RMSE for each model in the rmse_results dictionary and prints the best model along with its associated RMSE value. This process facilitates the comparison of model performance based on RMSE metrics and highlights the top-performing model in terms of predictive accuracy.

```
# Picking the best model with the lowest RMSE
best_model = min(rmse_results, key=rmse_results.get)
best_rmse = rmse_results[best_model]

print("RMSE for each model:")
for model, rmse in rmse_results.items(): print(f"{model}:
    {rmse}")

print(f"\nThe best model based on RMSE is {best_model} with RMSE:
{best_rmse}")
```

3. Predictions and Tensorflow Conversion for Test Set

   Finally, we ran predictions based on the best model instance.

```
# Converting into a TensorFlow dataset, training model and making predictions test_df
= tfdf.keras.pd_dataframe_to_tf_dataset(test_data, task = tfdf.keras.Task.REGRESSION)


preds = best_model_instance.predict(test_df)
```

Results and Reports:

RMSE for GBM: 24998

RMSE for Random Forest: 23813

References:

- Pandas Documentation
- Tensorflow for Python
- Keras API
- Sci-Kit Learn - Supervised Learning
- Tensorflow - Decision Forests