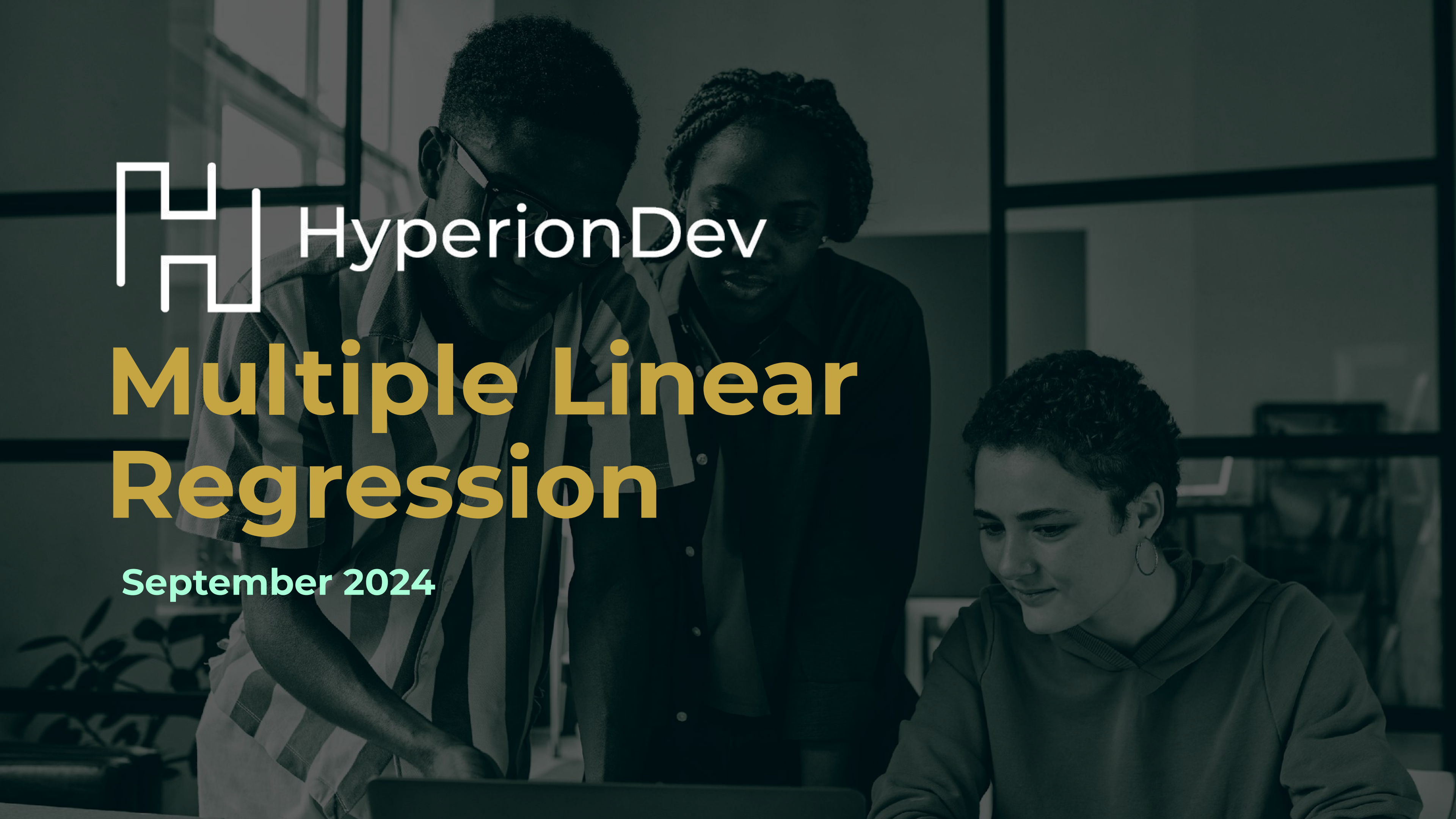


HyperionDev

Multiple Linear Regression

September 2024

Data Science Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Data Science Session Housekeeping cont.

- For all **non-academic questions**, please submit a query:
  www.hyperiondev.com/support

- Report a **safeguarding** incident:
  www.hyperiondev.com/safeguardreporting

- We would love your **feedback** on lectures: **[Feedback on Lectures]()**

Problem Statements

Imagine you are working on a project to predict house prices based on various factors such as square footage, number of bedrooms, and proximity to amenities. A Simple Linear Regression might tell you how house prices change with square footage alone, but it won't account for the influence of other important factors.

How can we alter our model to include the influence of other factors?
How does it affect the accuracy of the model?

Learning Outcomes

- **Explain** the concept of Multiple Linear Regression, its mathematical formulation, and its role in predictive modelling, including the assumptions and limitations of the model.
- **Implement** a Multiple Linear Regression model in Python using sklearn, including the steps of data preprocessing, feature scaling, model training, and making predictions.
- **Evaluate** the performance of an MLR model using metrics such as mean squared error, R2 score, and mean absolute error.

Why Multiple Linear Regression?

- In **Simple Linear Regression (SLR)**, a **single independent/predictor (x) variable** is used to model the **dependent/response/target variable (y).**

- In various cases, the response variable is affected by more than one predictor variable.

- **Multiple Linear Regression (MLR)** algorithm is used which models the **linear relationship** between **a single dependent continuous variable** and **more than one independent variable.**

- **Example:** Prediction of CO_2 emission based on engine size, with MLR more variables, like the weight and number of cylinders in the car, we can make the prediction more accurate.

Multiple Linear Regression

Extension of simple linear regression, uses **several explanatory (independent) variables** (x_i) to predict the outcome of **one response (dependent) variable** (y).

$$\boxed{y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_n x_n + \varepsilon}$$

y = **Output/Response/Dependent** variable

$x_1, x_2, x_3, \dots, x_n$ = Various **Feature/Explanatory/Independent** variables

β_0 = y-intercept (constant term)

$\beta_1, \beta_2, \beta_3, \dots, \beta_n$ = slope coefficient for each explanatory variable

ε = Model's **error** term (also called **residuals**)

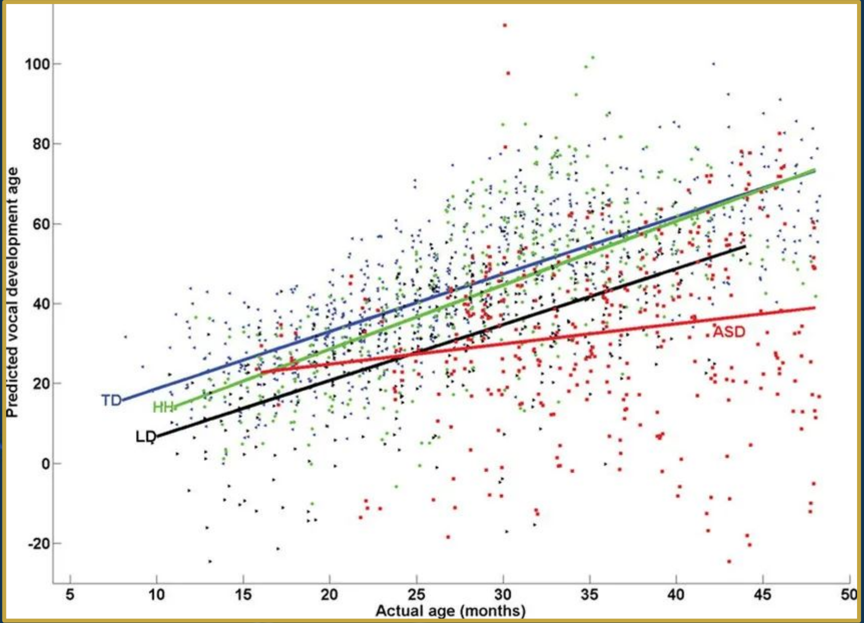
HyperionDev

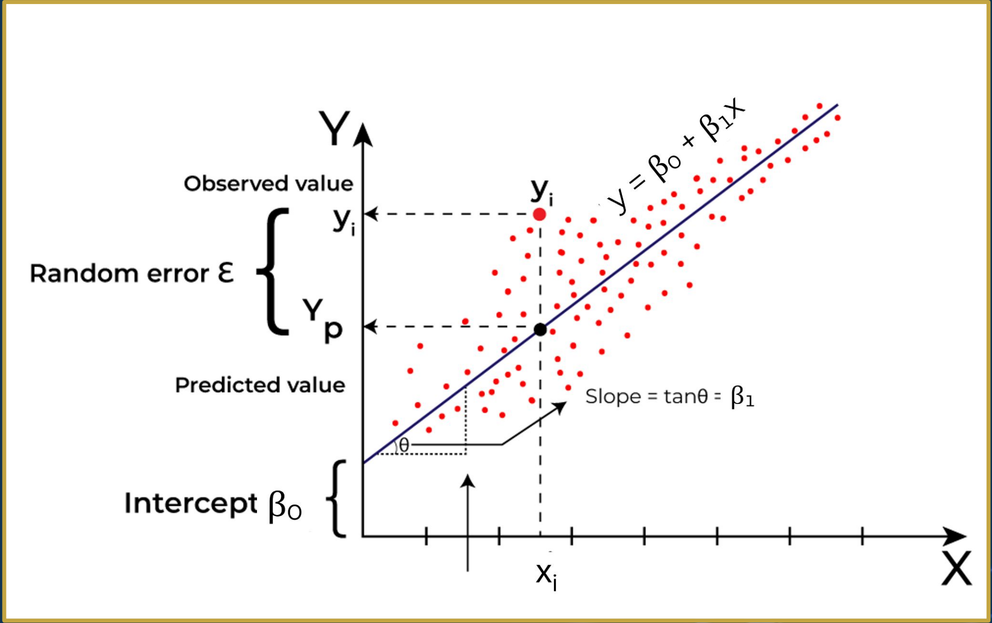
Multiple Linear Regression: Assumptions and Limitations

- **Linear relationship** between dependent and each independent variables.
- MLR assumes **little or no multicollinearity** (**correlation** between the **independent variable**) in data. If two independent variables are too highly correlated, then only one of them should be used in the regression model.
- Homogeneity of variance (**homoscedasticity**), size of **residuals** does not change significantly across values of independent variables.
- Regression **residuals** must be **normally distributed**.

Similar to SLR

Multiple Linear Regression





HyperionDev

Implementing MLR

scikit-learn (sklearn)

HyperionDev

scikit-learn

- **scikit-learn** is a popular Python library for machine learning.
- It provides simple and efficient tools for data analysis and modelling.

```
from sklearn.linear_model import LinearRegression
```

Multiple Linear Regression Example

- A linear approach to modelling the relationship between a scalar response and one or more explanatory variables.

- One use case is **predicting housing prices** based on various features such as size, location, and number of bedrooms.

Importing the dataset

Predicting housing prices based on various features such as size, location, and number of bedrooms.

```
#Data loading and manipulation
import numpy as np
import pandas as pd
#Data visualisation
import seaborn as sns
import matplotlib.pyplot as plt

# Import the dataset
df = pd.read_csv("housing.csv")
```

Checking the dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

SLR: Predicting house prices based on the **size of the house.**

MLR: Predicting house prices based on the **size of the house, number of bedrooms**, and **location**.

Feature Selection

Identifying which **features** most significantly influence the prediction i.e. housing prices. Preprocess data by **dropping irrelevant features**.

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386

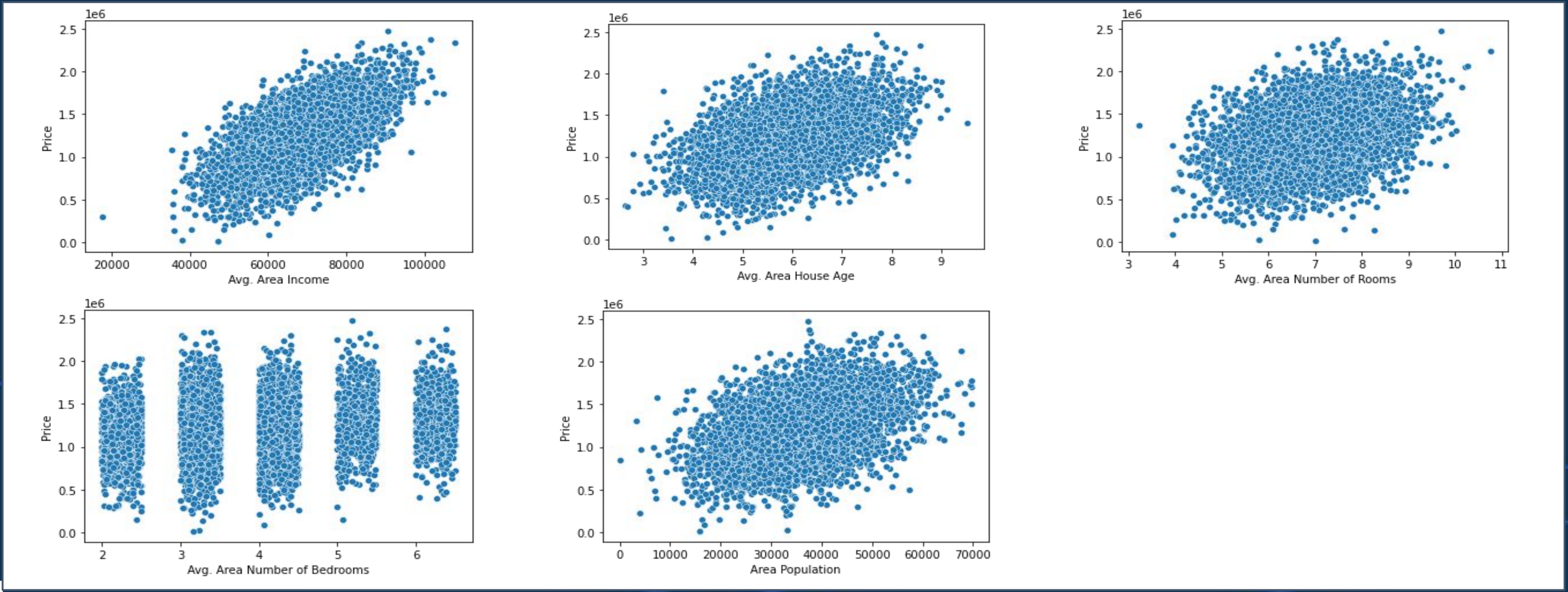
```python
# Drop the Address field as it's textual and not useful for regression without further processing
df = df.drop('Address', axis=1)
```

```python
# Feature matrix and target vector
X_feature = df.drop('Price', axis=1)  # We assume 'Price' is the column we want to predict
y_target = df['Price']
```

HyperionDev

Feature correlations

```
for col in X_feature.columns:
    sns.scatterplot(data=X_feature, x=X_feature[col], y=y_target)
```



Feature correlations


```
sns.heatmap(df.corr(), annot = True, cmap="YlGnBu")
```

Lack of correlation between Avg. Area Number of Bedrooms and the Price, we can drop this feature.

HyperionDev

```
# Updating Feature matrix by dropping Avg. Area Number of Bedrooms
X_feature = X_feature.drop('Avg. Area Number of Bedrooms', axis=1)
```

Multiple Linear Regression

- **LinearRegression()** method from **sklearn** to create a linear regression object.

- Method **fit(),** takes **independent (X_feature)** and **dependent (y_target)** values as parameters, fills regression object with data that describes the relationship.

- **Predict house price** based on given feature values, e.g., mean feature values

```
from sklearn.linear_model import LinearRegression

regr = LinearRegression()
regr.fit(X_feature, y_target)

#Predict the Price a house for
#the average values of the different features
averages = [X_feature[cols].mean() for cols in X_feature]
predicted_price = regr.predict([averages])
print(predicted_price)
print(regr.coef_)
#Output: [1232072.65414531]
#[2.15827436e+01 1.65657872e+05 1.21598165e+05 1.51961198e+01]
```

Price of house if the **average values of each feature** is taken = 1232072.65414531

If the average **income/house age/no. of rooms/area population** increase by 1 unit, the **Price** increases by the respective no. of units.

Dataset Splitting

train-test-split

HyperionDev

Dataset Splitting

- Aim is to get a model to predict the value of the price of the house.

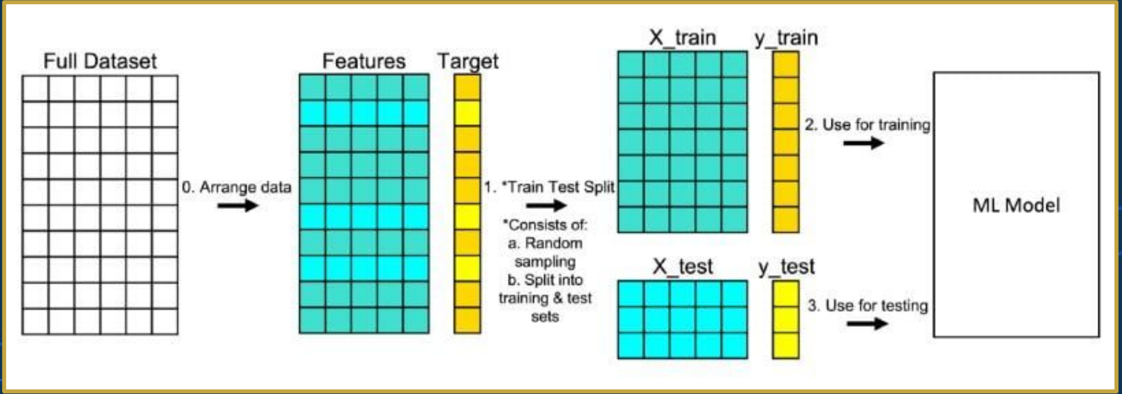
- **Train test split** is a model validation procedure that allows to simulate how a **model would perform on new/unseen data.**

- Divide the data into **training** and **testing** sets (sometimes a **validation set** too) for model evaluation.

Training set (to fit model)

Testing set (to evaluate model)

Ratio 80:20, 70:30, or 90:10



Source: https://builtin.com/data-science/train-test-split

HyperionDev

Data splitting, fitting model and predicting

```python
from sklearn.model_selection import train_test_split

# Split the dataset into training and testing sets
# Use the same random seed for learning purposes to get the same result
X_train, X_test, y_train, y_test = train_test_split(X_feature, y_target, test_size=0.3, random_state=42)
```

```python
# Initialize the multiple regression model
multi_reg_model = LinearRegression()
```

```python
# Fit the model to the training data
multi_reg_model.fit(X_train, y_train)
```

```python
# Predict the housing prices on the test set
y_pred = multi_reg_model.predict(X_test)
```

HyperionDev

Performance of model

Mean Squared Error, R^2 score, Mean Absolute Error

Evaluation Metrics

- **Mean Squared Error (MSE):**
  - MSE measures the **average squared difference** between the **predicted** and **actual** values.
  - **Root mean squared error (RMSE)** is root of MSE.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_p - y_a)^2$$

- **Mean Absolute Error (MAE)**
  - Measure of sum of **absolute errors** between **predicted** and **actual** values.

A **lower MSE and MAE** indicates better model performance.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_p - y_a|$$

Evaluation Metrics

- **R-squared (R^2) score** (coefficient of determination):
  - R^2 represents the **proportion of variance** in the target (dependent) variable that can be **explained by the independent variables/model.**
  - An R^2 value **closer to 1** indicates a better fit of the model to the data.
  - A value of 0 indicates that the model explains none of the variability of the response data around its mean.
  - A value of 1 indicates that the model explains all the variability of the response data around its mean.

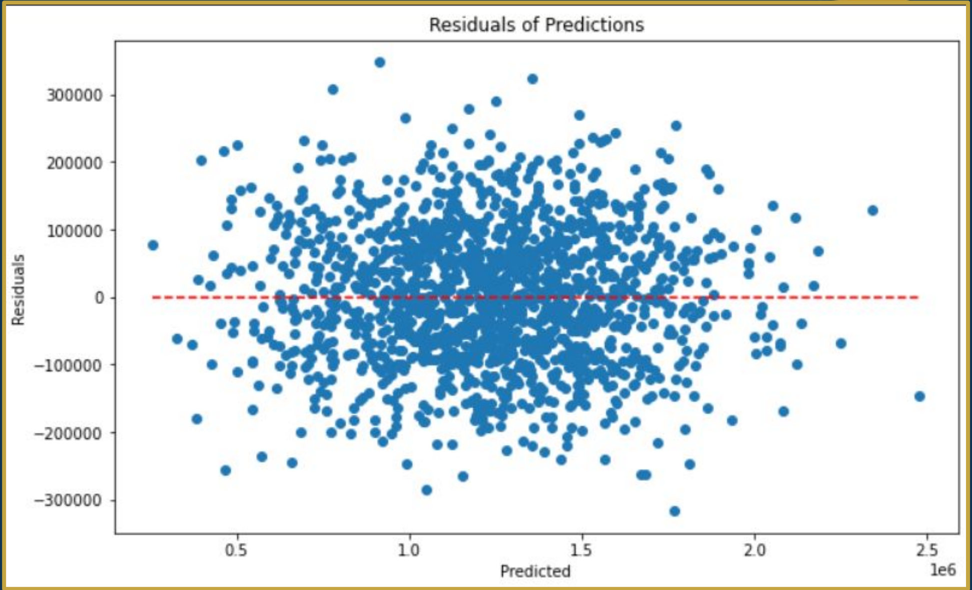
Evaluation Metrics

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
# Calculate the performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
```

```
Mean Squared Error: 10062092567.349297
R-squared: 0.9147354891150795
Mean Absolute Error: 81116.43359989364
```

Here we see quite a **high MSE and MAE (scale-variant metrics)**, although it also has **R^2 (scale-invariant metric)** value very **close to 1**, indicating that much of the variance can be explained by the current modeling - meaning it could potentially have a high goodness of fit.

Evaluation Metrics



One way to verify that our predictions are close to the actual values is plotting them against each other. If the line is straight it indicates a very good model.

Another valuable graph is looking at the residuals, which should be random for the model to be fitted well. In this case we can't see a clear pattern, suggesting a good model.

HyperionDev

Feature Scaling

Normalisation
Standardisation

HyperionDev

Feature Scaling

- Data columns have different values and measurement units, difficult to compare, e.g. What is kgs compared to metres? Or altitude compared to time?

- **Feature scaling - normalisation** and **standardisation**, involves transforming the data on the same scale
  - to compare easily,
  - more suitable for modeling,
  - to build accurate and effective machine learning models,
  - help improve model performance,
  - reduce the impact of outliers.

Feature Scaling

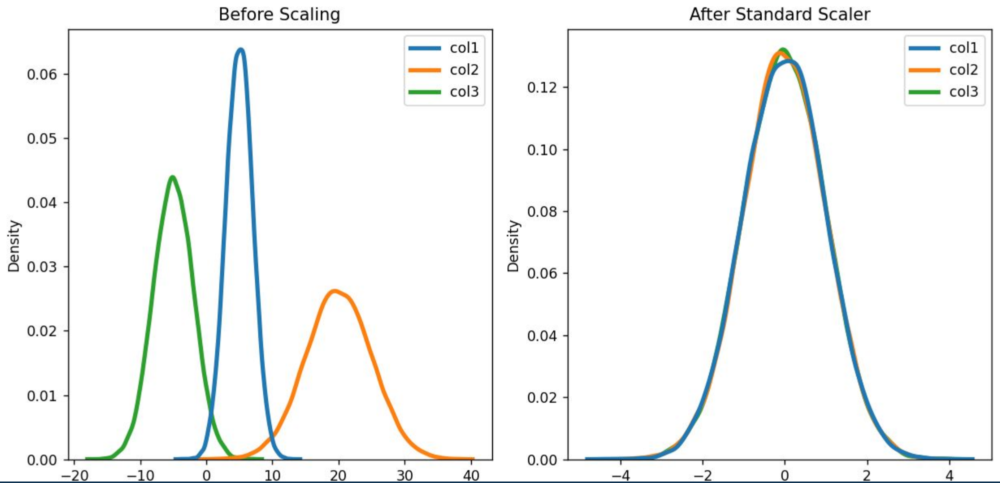
Normalisation	Standardisation
Rescales values to a range between 0 and 1	Centers data around mean and scales to standard deviation of 1
Useful when data distribution is unknown or **not Gaussian**	Useful with **Gaussian** data distribution
Sensitive to outliers	**Less sensitive to outliers**
Retains shape of original distribution	Changes shape of original distribution
May not preserve relationships between data points	Preserves relationships between data points
MinMaxScaler() (x – min)/(max – min)	**StandardScaler()** (x – mean)/standard deviation

HyperionDev

Feature Scaling examples

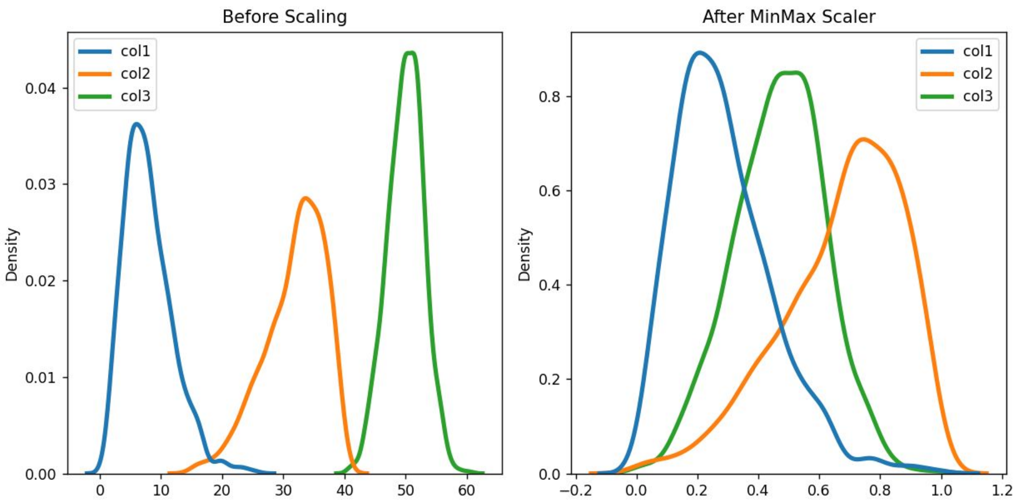
StandardScaler

Gaussian distributions



MinMaxScaler

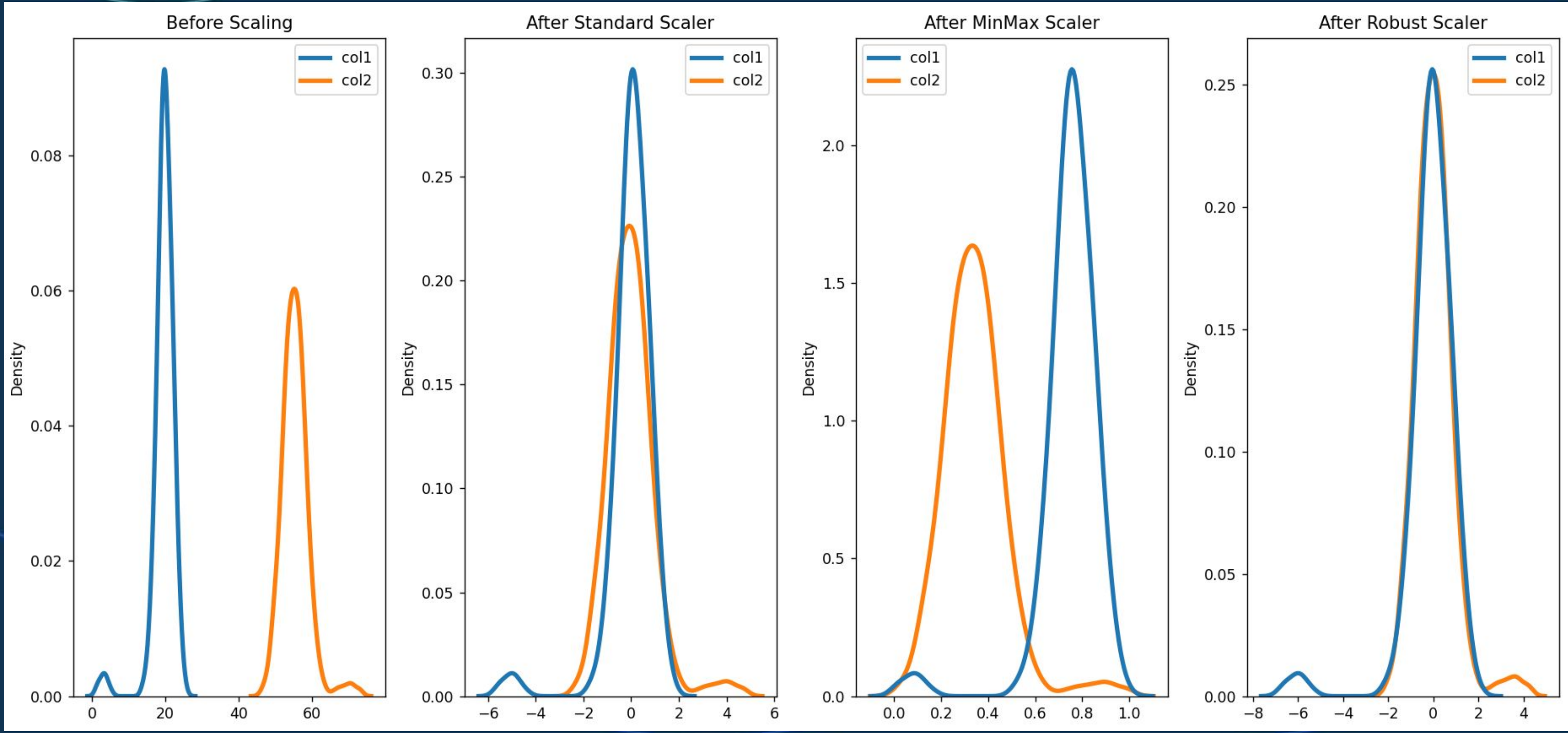
Non-Gaussian distributions



HyperionDev

Feature Scaling examples

Gaussian with outliers



HyperionDev

Feature Scaling

On housing dataset

```
from sklearn.preprocessing import StandardScaler
# Fit the scaler on train data
sc = StandardScaler() #MinMaxScaler() would be used if the data was not Gaussian
sc.fit(X_train)

# Apply the scaler on train and test data
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
```

HyperionDev

Normalising the data

```
#Normalizing numeric data
new_df = (df - df.mean()) / df.std()
new_df.head()
```

Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
1.028557	-0.296897	0.021272	0.088053	-1.317467	-0.490032
1.000708	0.025899	-0.255481	-0.722229	0.403959	0.775431
-0.684561	-0.112292	1.516092	0.930747	0.072403	-0.490162
-0.491450	1.221450	-1.392938	-0.584481	-0.186716	0.080835
-0.806992	-0.944739	0.846657	0.201493	-0.988289	-1.702348

```
Mean Squared Error: 0.08069553676689753
R-squared: 0.9147354891150797
Mean Absolute Error: 0.22971505100062556
```

HyperionDev

Low MSE and MAE, R^2 score ~ 1

Questions and Answers

Thank you for attending