Introduction to Machine Learning in Python – Workshop 3

Presented by Jiajia Li

07.08.2024

Australian National University

Agenda

01	Recap
02	Evaluation - Regression
03	Residual Analysis
04	Regression Models
05	Exercise
06	
07	

Recap – Multiple logistic regression

Demonstrate on how to fit all 13 features to the logistic regression model in Jupyter Notebook.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression

x = cohort_enc.drop('actualhospitalmortality_enc', axis=1)
y = cohort_enc['actualhospitalmortality_enc']

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=.7, random_state=42)

x_train = x_train.fillna(x_train.median())
x_test = x_test.fillna(x_train.median())

scaler = MinMaxScaler()
scaler.fit(x_train)
x_apache_train = scaler.transform(x_train)
x_apache_test = scaler.transform(x_test)

model = LogisticRegression()
model = model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

Recap – Multiple logistic regression

A warning message popped up:

```
/home/jiajia/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

This indicates that the logistic regression algorithm using the ‘lbfgs’ solver did not converge within the allowed number of iterations.

What does converge mean here?
What is ‘lbfgs’ solver?

Recap – Multiple logistic regression

Convergence in the context of logistic regression means that the algorithm has successfully found the optimal parameters that minimise the loss function.

Convergence occurs when further updates result in little or no improvement in the loss function, indicating that the algorithm has found the best set of parameters for the given data.

'lbfgs' stands for **Limited-memory Broyden-Fletcher-Goldfarb-Shanno**. It is an optimisation algorithm that is particularly well-suited for problems with a large number of parameters.

'lbfgs' is the default solver/optimisation method of 'LogisticRegression()' function.

Recap – Multiple logistic regression

Let's have a look of the default settings of function 'LogisticRegression()':

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *,
dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1,
class_weight=None, random_state=None, solver='lbfgs', max_iter=100,
multi_class='deprecated', verbose=0, warm_start=False, n_jobs=None,
l1_ratio=None) #                                                    [sourc
```

This image was cut from the sklearn documentations. We can see that the default solver is 'lbfgs', and the default maximum iterations is '100'.

To increase the iterations, we need to change the setting for 'max_iter'.

Recap – Multiple logistic regression

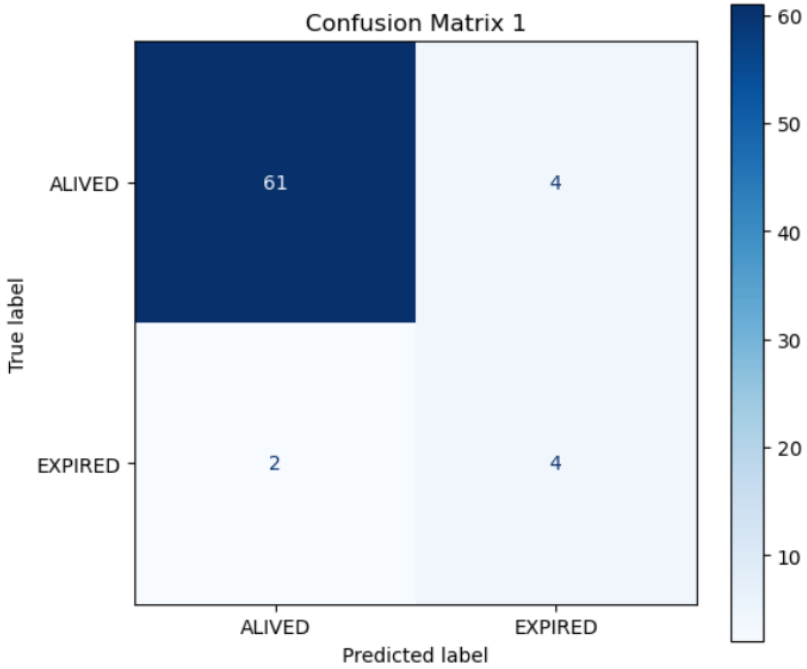
Let's change 'max_iter' to 1000, and run the code again:

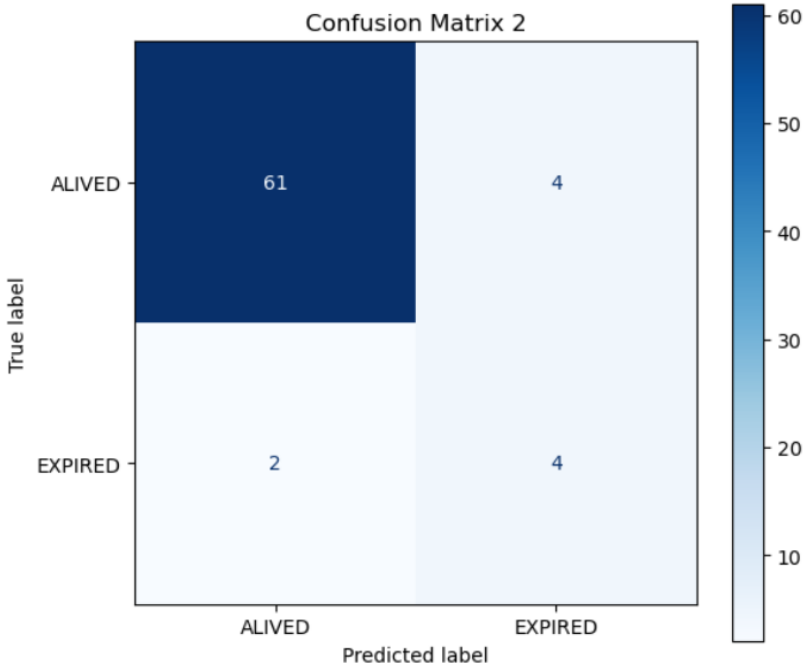
```
model = LogisticRegression(max_iter=1000)
model = model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

This time, we didn't get the warning message, it means that our model has converged successfully.

How can we inspect the results of this new model?

Exercise – Compare two models by creating confusion matrices





Melbourne Housing Snapshot

Let’s read in this file and have a glimpse of our data.

Target

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Bathroom	Car	Landsize	BuildingArea	YearBuilt	CouncilArea
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	3067.0	...	1.0	1.0	202.0	NaN	NaN	Yarra
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	3067.0	...	1.0	0.0	156.0	79.0	1900.0	Yarra
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	3067.0	...	2.0	0.0	134.0	150.0	1900.0	Yarra
3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin	4/03/2017	2.5	3067.0	...	2.0	1.0	94.0	NaN	NaN	Yarra
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	2.5	3067.0	...	1.0	2.0	120.0	142.0	2014.0	Yarra

Housing – check missing values

```
Suburb            0
Address           0
Rooms             0
Type              0
Price             0
Method            0
SellerG           0
Date              0
Distance          0
Postcode          0
Bedroom2          0
Bathroom          0
Car              62
Landsize          0
BuildingArea   6450
YearBuilt      5375
CouncilArea    1369
Lattitude         0
Longtitude        0
Regionname        0
Propertycount     0
dtype: int64
```

We can see that there are 4 columns have missing values.

- **Car** – important variable, impute it with median.
- **BuildingArea** – important variable, impute it with median.
- **YearBuilt** – important variable, impute it with median.
- **CouncilArea** – equivalent to other location indicators, remove this column.

Handling missing data is crucial in machine learning, and there are many different strategies to do it. We won't cover it in detail for now.

Housing – drop columns

```
Suburb          object
Address         object
Rooms            int64
Type            object
Price          float64
Method          object
SellerG         object
Date            object
Distance       float64
Postcode       float64
Bedroom2       float64
Bathroom       float64
Car            float64
Landsize       float64
BuildingArea   float64
YearBuilt      float64
CouncilArea     object
Lattitude      float64
Longtitude     float64
Regionname      object
Propertycount  float64
dtype: object
```

A few columns have similar data which shows the location of the property, we can remove them.

Which columns should we remove?

- Address
- Distance
- Postcode
- CouncilArea
- Lattitude
- Longtitude
- Regionname
- Propertycount

Housing – data encoding

```
Suburb          object
Rooms            int64
Type            object
Price          float64
Method          object
SellerG         object
Date            object
Postcode       float64
Bedroom2       float64
Bathroom       float64
Car            float64
Landsize       float64
BuildingArea   float64
YearBuilt      float64
```

The variables that are shown as ‘object’ here, it indicates categorical variables. Encode columns:

- Suburb
- Type
- Method
- SellerG
- Date

Column ‘Date’ is time series data, we will use it as categorical data for now.

Housing – Fit to model

After a quick cleaning, we can now fit our data to a linear regression model.

Evaluation for regression tasks

Metrics for evaluation:

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R-squared (R^2)
- Adjusted R-squared (Adjusted R^2)

Residual Analysis.

Evaluation for regression tasks

Mean Absolute Error (MAE): MAE is the average of the absolute differences between the predicted values and the actual values.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

A lower MAE indicates better model performance.

MAE is in the same unit as the original data, which makes it easy to interpret.

Evaluation for regression tasks

Mean Squared Error (MSE): MSE is the average of the squares of the differences between the predicted values and the actual values. By squaring the errors, MSE gives more weight to larger errors, making it sensitive to outliers.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

A lower MSE indicates better model performance.

MSE is useful for capturing the variance of the errors, but its unit is the square of the original data's unit, which can make it less interpretable.

Evaluation for regression tasks

Root Mean Squared Error (RMSE): RMSE is the square root of the average of the squared differences between the predicted values and the actual values. It combines the advantages of both MAE and MSE by providing a measure that is sensitive to large errors while retaining the same unit as the original data.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

A lower RMSE indicates better model performance.

RMSE is in the same unit as the original data, making it easy to interpret.

Because it penalises larger errors more than MAE, it can be more sensitive to outliers.

Evaluation for regression tasks

R-squared (R^2), also known as the coefficient of determination, is a statistical measure that represents the proportion of the variance for the dependent variable that's explained by the independent variables.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

For example, an R^2 value of 0.8 means that 80% of the variance in the dependent variable is explained by the independent variable(s). And other 20% of the variance is explained by other things which is missing in our model.

It provides an indication of the goodness of fit of the model. A higher R^2 value indicates a better fit of the model to the data.

However, a very high R^2 value may also indicate overfitting, especially in models with many predictors.

Evaluation for regression tasks

Adjusted R-squared: to address the limitation related to the number of predictors, we use Adjusted R-squared. Adjusted R^2 modifies the R-squared value by adjusting for the number of predictors in the model.

$$\text{Adjusted } R^2 = 1 - \left(\frac{1-R^2}{n-k-1}\right)(n-1)$$

- n is the number of observations.
- k is the number of predictors.

Adjusted R-squared can be lower than R-squared, and it can decrease if adding a new predictor does not improve the model significantly.

Code implementation in Python

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

n = len(y_test)
k = x_train.shape[1]
adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - k - 1)
```

```
Mean Absolute Error (MAE): 343753.722049045
Mean Squared Error (MSE): 257535208129.36462
Root Mean Squared Error (RMSE): 507479.26866953354
R-squared (R2): 0.386808770720491
Adjusted R-squared: 0.3849968291417286
```

The R^2 and Adjusted R^2 of our linear regression model are around 0.38, indicating a not-so-good regression fit.

Residual Analysis

Residuals are the differences between observed and predicted values.

```
residuals = y_test - y_pred
```

Analysing the residuals can help diagnose issues with the model, such as non-linearity, outliers, and heteroscedasticity (non-constant variance of residuals).

There are several analyses we can do:

- Residual vs. Fitted Plot
- Q-Q Plot
- Histogram of Residuals
- Residual vs. Predictor Plot

Residual Analysis – Heteroscedasticity

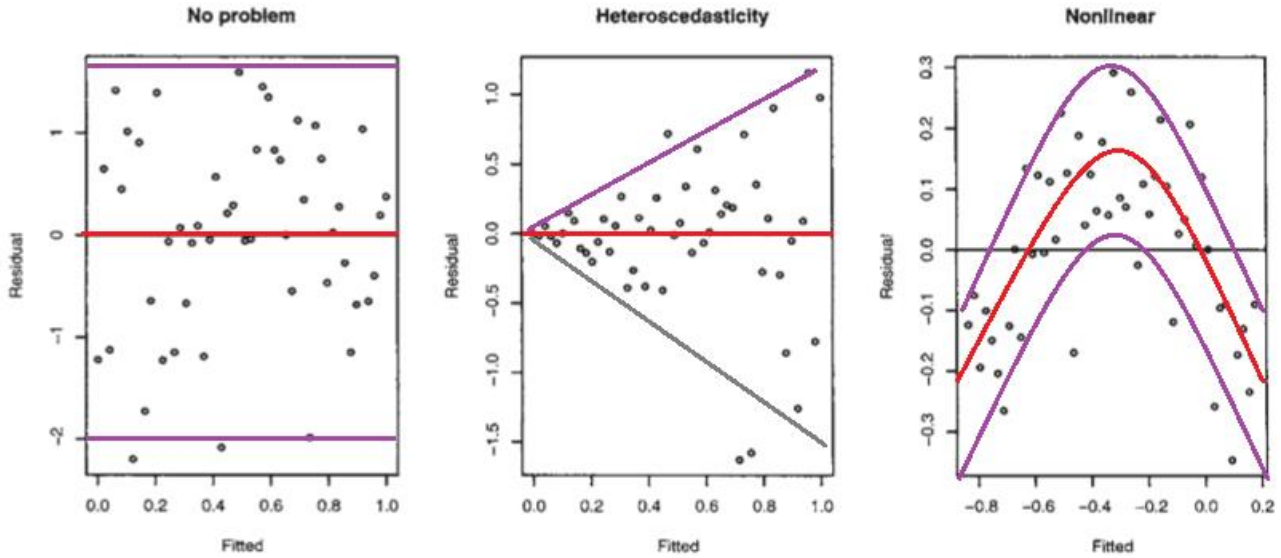
Heteroscedasticity occurs when the variance of the residuals (the differences between the observed and predicted values) is not constant across all levels of the independent variables.

In other words, the spread or scatter of the residuals varies across different values of the independent variables.

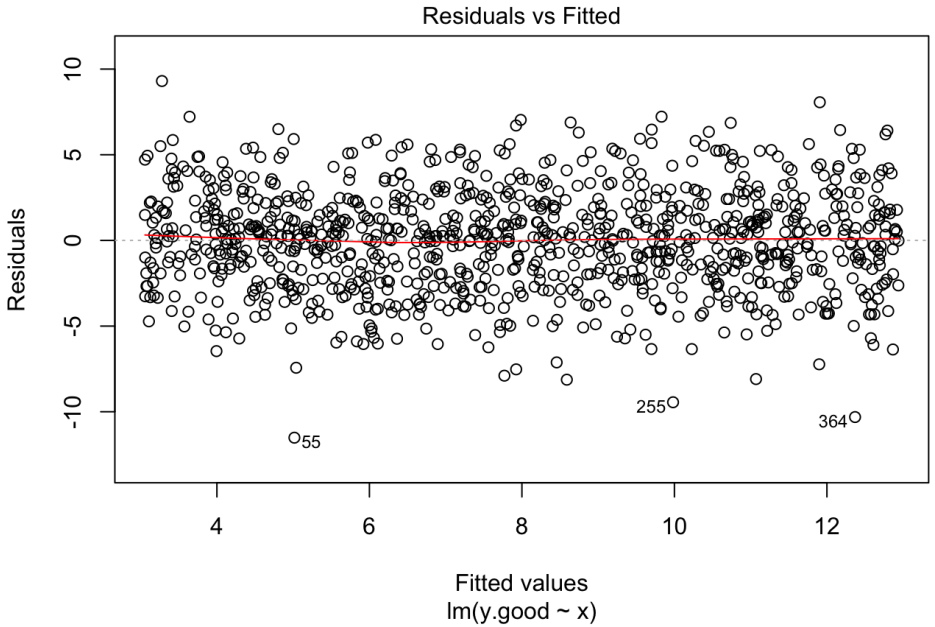
This is a violation of one of the key assumptions of linear regression, which assumes homoscedasticity (constant variance of residuals).

Residual Analysis – Residual vs. Fitted Plot

This plot helps to check the assumption of linearity and homoscedasticity (constant variance).



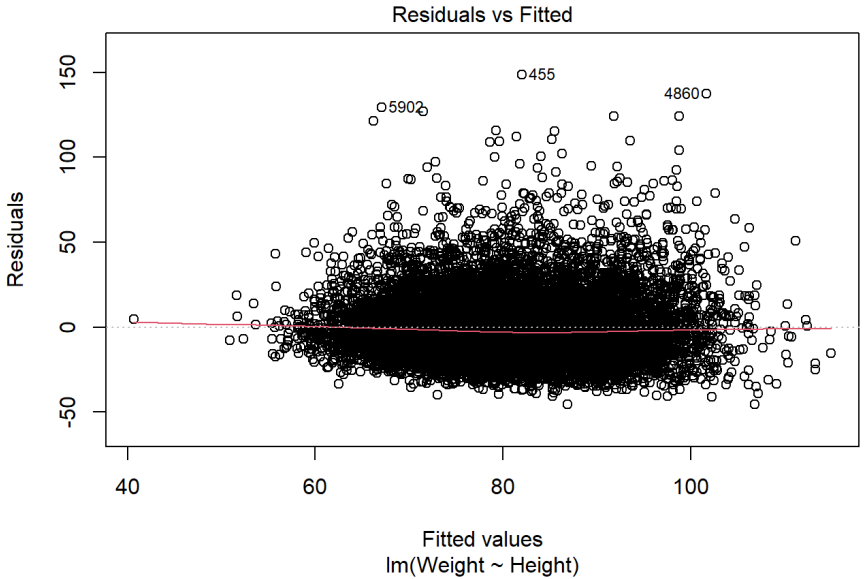
Residual Analysis – Residual vs. Fitted Plot



This plot indicates a good fit.

The residuals are randomly scattered with no clear pattern.

Residual Analysis – Residual vs. Fitted Plot

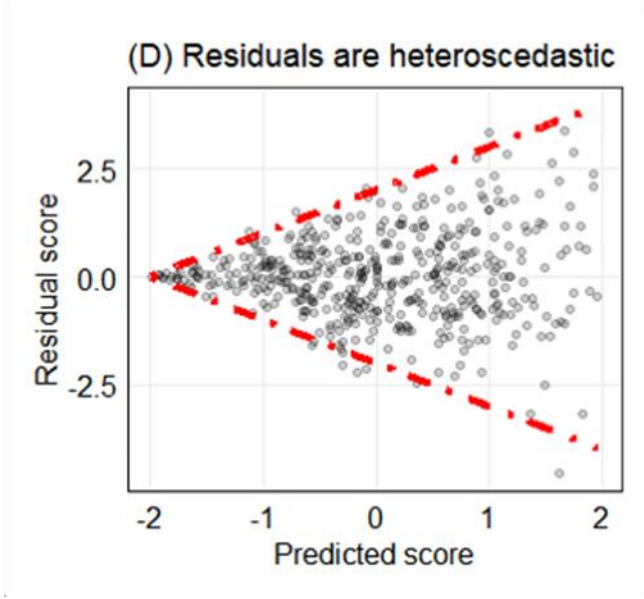


This plot also indicates a good fit.

The residuals are randomly scattered with no clear pattern.

There are a few outliers.

Residual Analysis – Residual vs. Fitted Plot

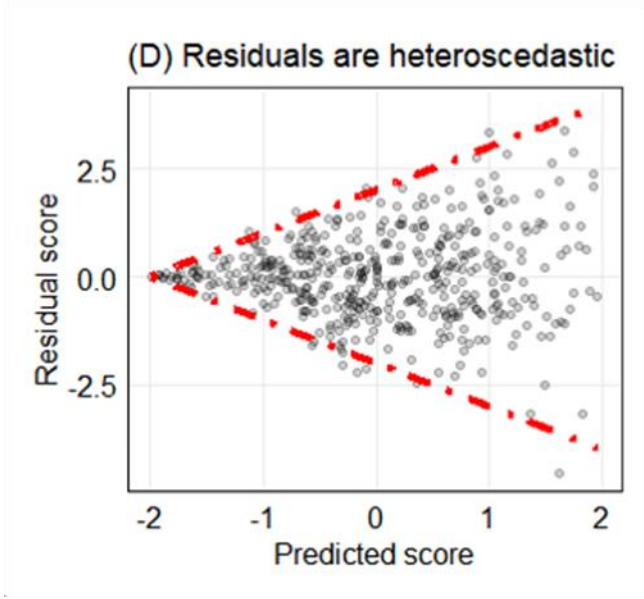


This plot shows heteroscedasticity.

On the left, for low fitted values, the residuals are small, meaning that here the model prediction is very good. Low predicted values are very close to the actual scores.

But for high fitted values the prediction quality is poor, with many large residuals.

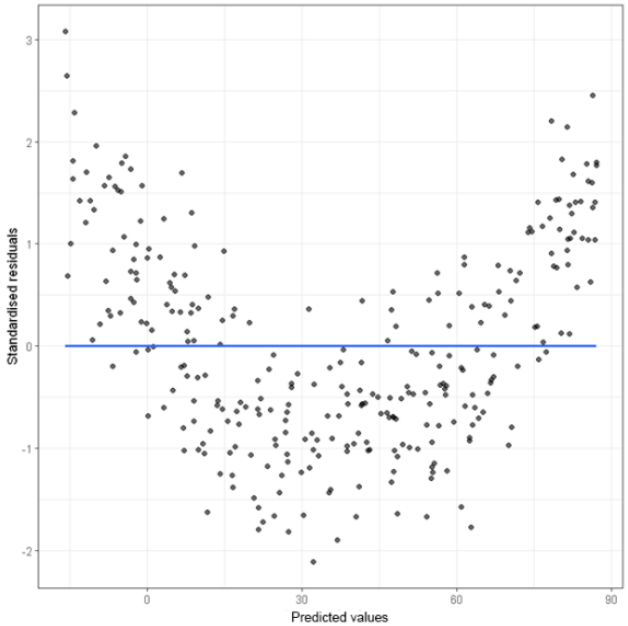
Residual Analysis – Residual vs. Fitted Plot



Why there is heteroscedasticity?

- Important variables that influence the dependent variable are missing from the model.
- The relationship between the independent and dependent variable is not correctly specified (e.g., a linear model when the relationship is actually quadratic).
- Outliers in the data can cause residuals to have non-constant variance.

Residual Analysis – Residual vs. Fitted Plot



This plot shows non-linearity.

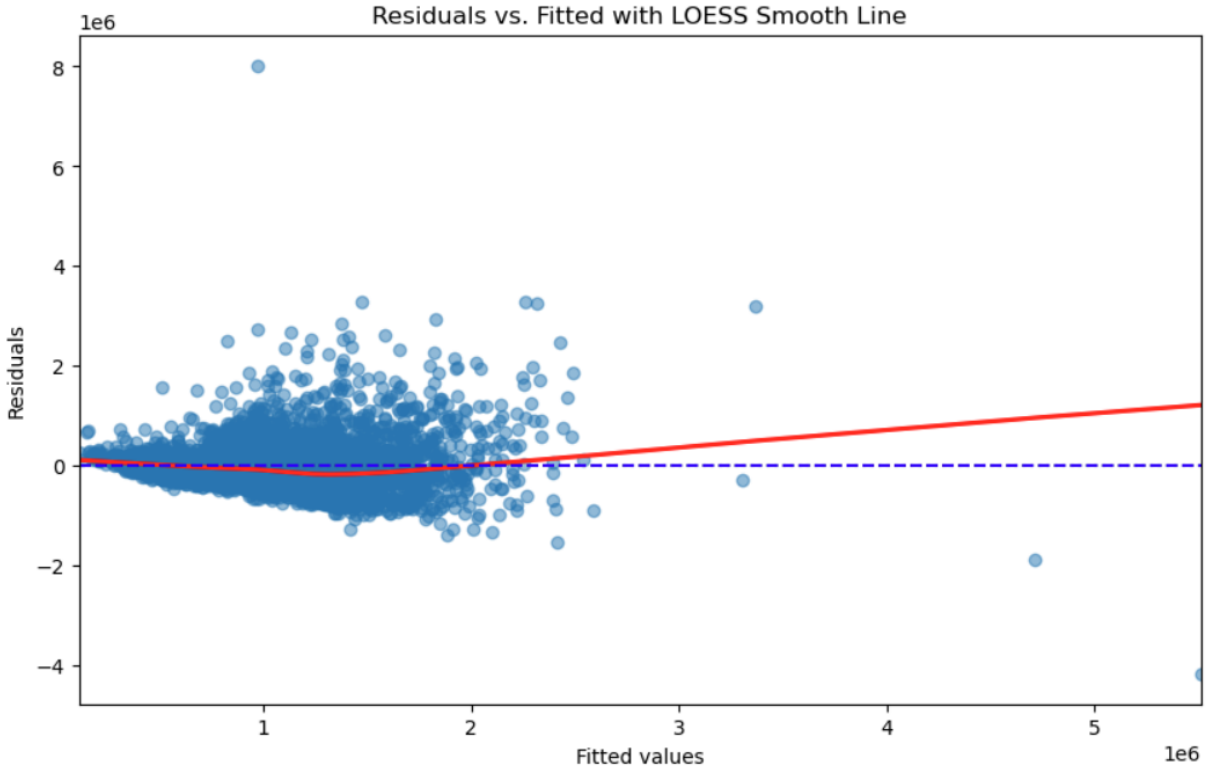
Nonlinearity in the context of regression analysis refers to a situation where the relationship between the independent variables and the dependent variable cannot be adequately described by a straight line.

Consider fit models that can capture complex relationships, such as decision trees, random forests, and neural networks.

Residual Analysis – Residual vs. Fitted Plot

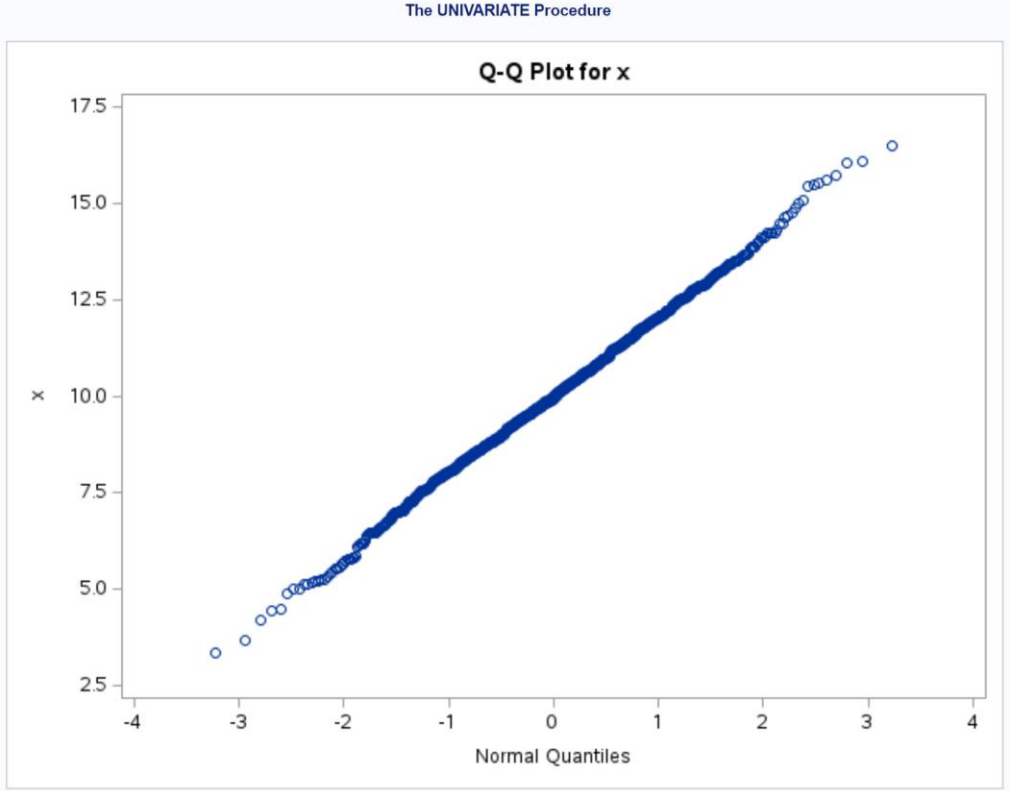
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.residplot(x=y_pred, y=residuals, lowess=True, scatter_kws={'alpha':0.5}, line_kws={'color':'red'})
plt.axhline(y=0, color='blue', linestyle='--')
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Fitted with LOESS Smooth Line')
plt.show()
```

Residual Analysis – Residual vs. Fitted Plot



What problems do we have for our model?

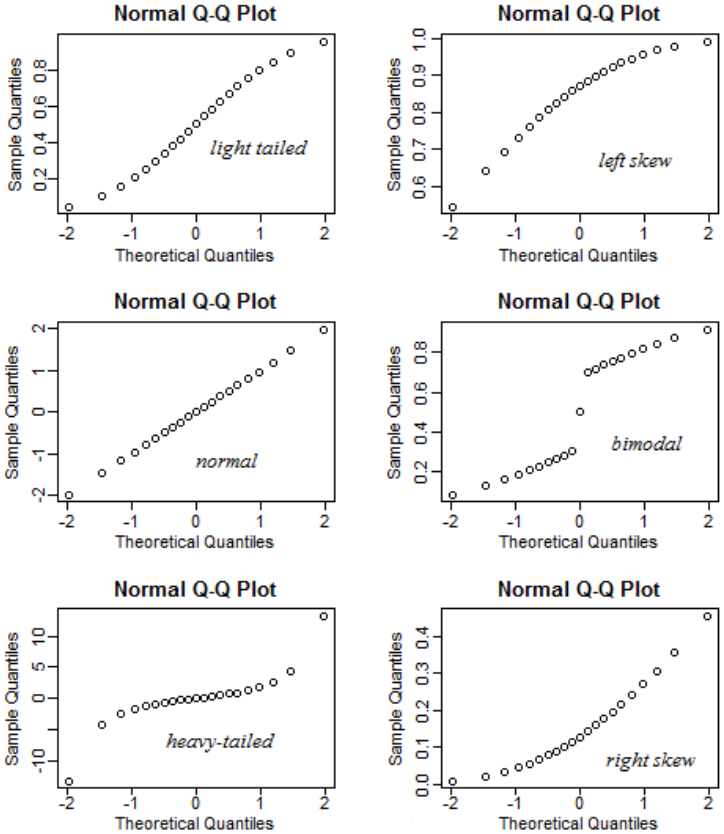
Residual Analysis Q-Q Plot



Creating a Q-Q plot for the residuals of a linear regression model helps assess whether the residuals follow a normal distribution, which is one of the assumptions of linear regression.

If the residuals are normally distributed, the points will lie approximately along the 45-degree reference line.

Residual Analysis Q-Q Plot



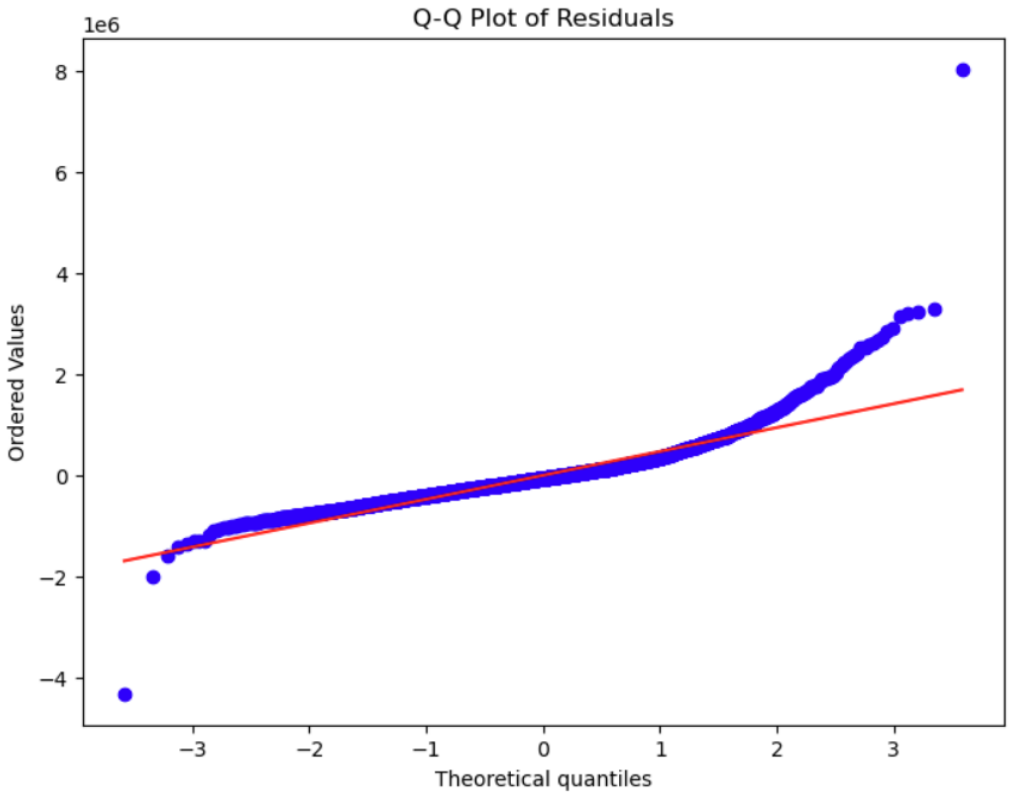
Normal: the normal distribution is symmetric, so it has no skew.

Light tailed: meaning that compared to the normal distribution there is little more data located at the extremes of the distribution and less data in the centre of the distribution.

Heavy tailed: meaning that compared to the normal distribution there is much more data located at the extremes of the distribution and less data in the centre of the distribution.

Bimodal: illustrate a bimodal distribution.

Residual Analysis Q-Q Plot




```python
import scipy.stats as stats
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
stats.probplot(residuals, dist="norm", plot=plt)
plt.title('Q-Q Plot of Residuals')
plt.show()
```

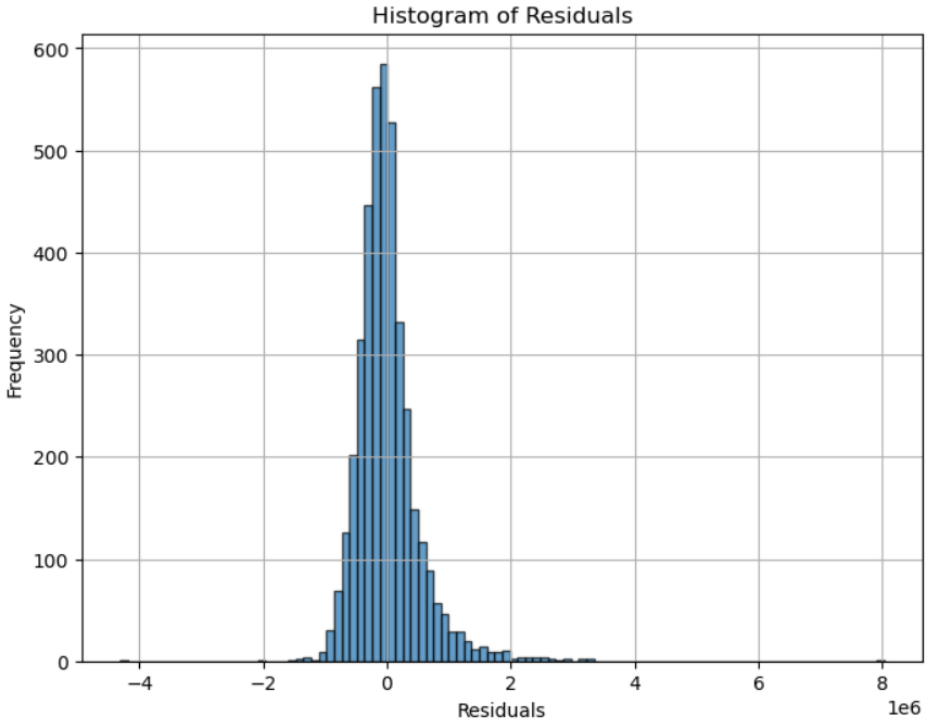
Our residuals are heavy tailed, meaning there is much more data located at the extremes of the distribution.

There are also two extreme outliers.

Residual Analysis – Histogram of residuals

The histogram of residuals should follow a normal distribution.

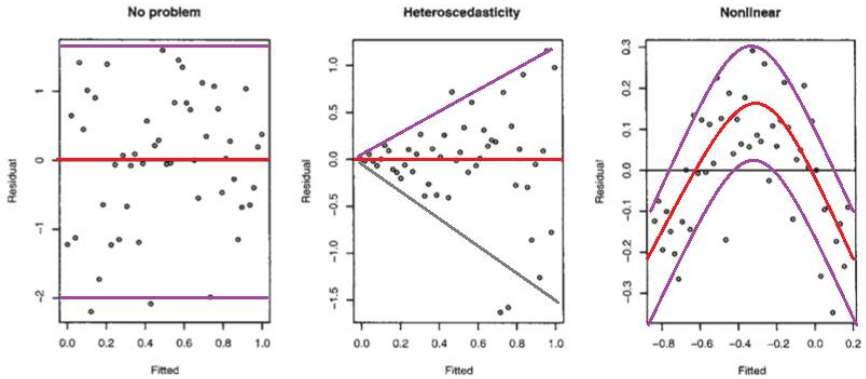
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.hist(residuals, bins=100, edgecolor='k', alpha=0.7)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



Residual Analysis – Residuals vs. Predictor Plot

A residuals vs. predictor plot is a diagnostic tool used to assess the fit and assumptions of a regression model. It plots the residuals from the regression model against the predictor variable.

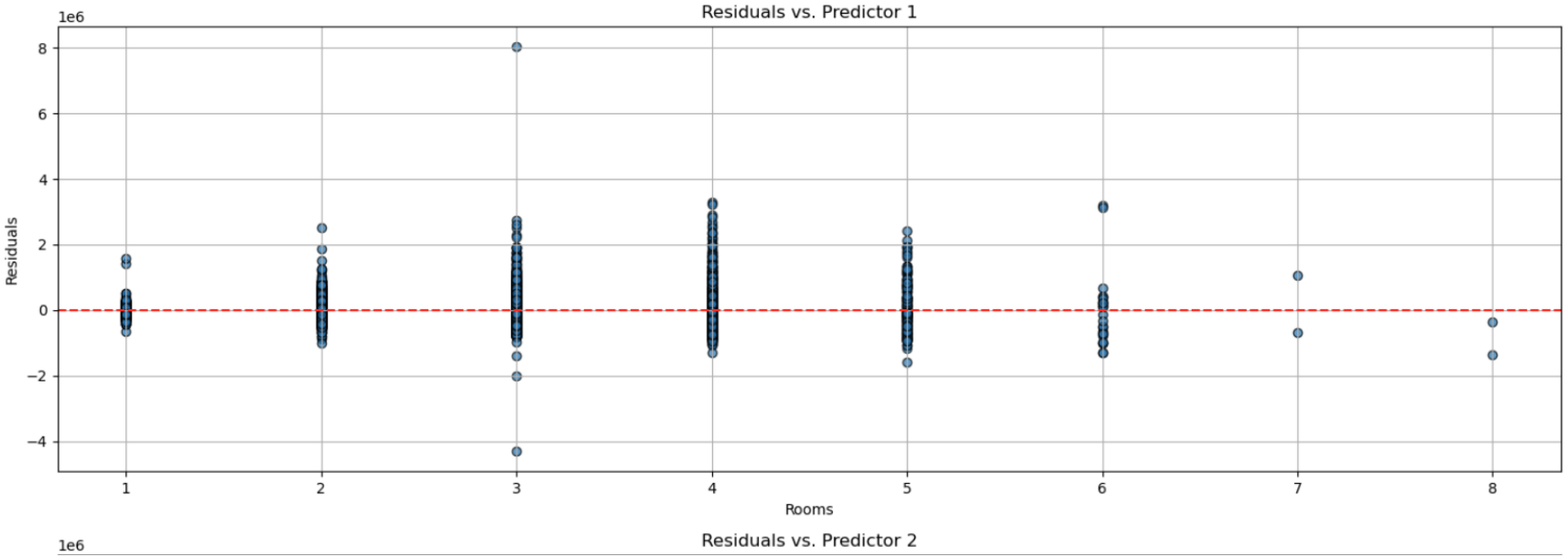
The way to interpret a residuals vs. predictor plot is the same as residuals vs. fitted plot. Residuals should be randomly scattered around zero with no clear pattern.



Residual Analysis – Residuals vs. Predictor Plot

```python
import matplotlib.pyplot as plt

num_predictors = x_test.shape[1]
plt.figure(figsize=(15, 5 * num_predictors))

for i in range(num_predictors):
    plt.subplot(num_predictors, 1, i + 1)
    plt.scatter(x_test.iloc[:, i], residuals, alpha=0.7, edgecolor='k')
    plt.axhline(0, color='r', linestyle='--')
    plt.title(f'Residuals vs. Predictor {i + 1}')
    plt.xlabel(x_test.columns[i])
    plt.ylabel('Residuals')
    plt.grid(True)

plt.tight_layout()
plt.show()
```

Residual Analysis – Residuals vs. Predictor Plot



Machine learning models for regression

There are several machine learning models commonly used for regression tasks, each with its own strengths and use cases. Here are some popular ones:

- Linear Regression
- Ridge Regression
- Lasso Regression
- Elastic Net
- Decision Tree Regression
- Random Forest Regression
- Gradient Boosting Regression
- Support Vector Regression
- Neural Networks

Ridge regression (L2 Regularisation)

Adds a penalty equal to the square of the magnitude of coefficients to the loss function, which helps prevent overfitting.

Use Case: when dealing with multicollinearity or when you need to stabilise the regression coefficients.

Pros: helps reduce overfitting, improves model generalisation.
Cons: can't reduce coefficients to zero, so less useful for feature selection.

```
from sklearn.linear_model import Ridge

ridge_model = Ridge(alpha=1.0)
ridge_model.fit(x_train, y_train)
```

Lasso Regression (L1 Regularization)

Adds a penalty equal to the absolute value of the magnitude of coefficients to the loss function, which can shrink some coefficients to zero.

Use Case: when feature selection is needed, or when you have many features.

Pros: performs both regularization and feature selection.
Cons: can be sensitive to the choice of regularization parameter.

```
from sklearn.linear_model import Lasso

lasso_model = Lasso(alpha=0.1)
lasso_model.fit(x_train, y_train)
```

Elastic Net

Combines L1 and L2 regularization penalties. It's useful when there are multiple features correlated with each other.

Use Case: When you need both regularization and feature selection.

Pros: Combines advantages of both Ridge and Lasso regression.
Cons: Requires tuning two hyperparameters.

```
from sklearn.linear_model import ElasticNet

alpha = 0.1
l1_ratio = 0.5
model = ElasticNet(alpha=alpha, l1_ratio=l1_ratio, random_state=0)
model.fit(x_train, y_train)
```

Decision Tree Regression

Uses a decision tree structure to model the relationship between variables. It splits the data into subsets based on feature values.

Use Case: When the relationship between variables is highly non-linear and complex.

Pros: Can capture non-linear relationships, easy to interpret.
Cons: Can overfit the data, especially with deep trees.

```
from sklearn.tree import DecisionTreeRegressor

tree_model = DecisionTreeRegressor()
tree_model.fit(x_train, y_train)
```

Random Forest Regression

An ensemble method that builds multiple decision trees and averages their predictions to improve performance and robustness.

Use Case: When you need a more robust model that reduces the risk of overfitting.

Pros: Reduces overfitting, handles non-linearities well.
Cons: Less interpretable, can be computationally intensive.

```
from sklearn.ensemble import RandomForestRegressor

rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(x_train, y_train)
```

Gradient Boosting Regression

Builds models sequentially, where each model attempts to correct the errors of the previous one. Examples include XGBoost, LightGBM, and CatBoost.

Use Case: When high predictive performance is needed, and the data is complex.

Pros: High predictive accuracy, handles non-linearities well.
Cons: Can be sensitive to hyperparameters, computationally intensive.

```
from sklearn.ensemble import GradientBoostingRegressor

gb_model = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_model.fit(x_train, y_train)

y_pred_gb = gb_model.predict(x_test)
```

```
import xgboost as xgb

xgb_model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, random_state=42)
xgb_model.fit(x_train, y_train)
```

Support Vector Regression (SVR)

Uses support vector machines to perform regression, aiming to fit the best line within a margin of tolerance.

Use Case: When you need a model that can handle non-linear relationships and is robust to outliers.

Pros: Effective in high-dimensional spaces, robust to overfitting.
Cons: Can be computationally expensive, especially with large datasets.

```
from sklearn.svm import SVR

svr_model = SVR(kernel='rbf', C=1.0, epsilon=0.2)
svr_model.fit(x_train, y_train)
```

Neural Networks

Uses layers of interconnected nodes to model complex relationships between features and targets.

Use Case: When dealing with very complex relationships and large datasets.

Pros: Highly flexible, can model very complex relationships.
Cons: Requires large amounts of data, can be computationally expensive, and harder to interpret.

```
from sklearn.neural_network import MLPRegressor

nn_model = MLPRegressor(hidden_layer_sizes=(100,), max_iter=1000, random_state=42)
nn_model.fit(x_train, y_train)
```

Overfitting

Overfitting in machine learning occurs when a model learns not just the underlying patterns in the training data but also the noise and random fluctuations.

This results in a model that performs very well on the training data but poorly on new, unseen data.

Essentially, the model becomes too complex and is too closely tied to the specifics of the training set, making it less generalisable.

Model Selection

When you fit the data to different models, you can experience the time difference to train each model. The more complex the model is, the more computational power it requires.

Not all of them will get a better result than simple linear regression.

There are more steps in the model selection process we can do to increase the goodness-of-fit of our model, including different ways to handle missing data, remove outliers, feature selection, hyperparameter tuning, and cross validation.

We will introduce some of it in our following workshops.

Exercise

1. Clean the Melbourne Housing dataset and fit it to a linear regression model.

2. Evaluate the fitted linear regression model by calculating MAE, MSE, RMSE, R-squared, and Adjusted R-squared.

3. Conducting residual analysis on the fitted linear regression model.

4. Fit the data to other regression models and calculate the RMSE and R-squared and compare them.

References

- OpenAI – ChatGPT
- Dan Becker – Handling Missing Values
- Suresh Kumar – PRODUCTION FUNCTION ANLYSIS Book Chapter
- Glen_b – Interpreting the residuals vs. fitted values plot for verifying the assumptions of a linear model
- Prof. Chouldechova – Regression diagnostic plots
- DHSC Analysts - R for Survey Analysis Chapter 11 Testing regression assumptions
- Radboud University – Linear regression

THANK YOU

Contact Us

Biological Data Science Institute
ANU College of Science

RN Robertson Building
46 Sullivans Creek Rd
Acton ACT 2601

jiajia.li1@anu.edu.au

Australian National University