VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

Applied Mathematics and Statistics

Project 03

Linear Regression

Student: 21127135 - Diep Huu Phuc
Class: 21CLC05
Instructors: Vu Quoc Hoang
Nguyen Van Quang Huy
Le Thanh Tung
Phan Thi Phuong Uyen

Ho Chi Minh City, August 2023

CONTENTS

Project 03: Linear Regression

Diep Huu Phuc [1★]
[1] *Faculty of Information Technology, VNUHCM-University of Science, Vietnam*
★ *Student ID: 21127135, GitHub: kru01*

Abstract

Often we want to make educated predictions based on already known information, this is a core idea of many supervised machine learning algorithms. Aside from Classification guessing dataset's class, another kind of supervised learning is Regression which projects continuous output variables derived from independent input variables. The most straightforward type of regression is Linear Regression which quantifies the linear connection between a target variable and one or many explanatory variables. In this document, I will employ linear regression to train, test, and build simple linear models.

1 INTRODUCTION

In statistics, **Linear Regression** is a linear approach for modelling the relationship between a scalar response (dependent variable) and one or more explanatory (independent) variables. The case of more than one explanatory variables is called **multiple linear regression**. Relationships, called **linear models**, are shaped using **linear predictor functions** whose unknown model parameters are estimated from the data (Wikipedia contributors, 2023a). One very common method for choosing such parameters is **Ordinary Least Squares (OLS)**, which works by minimizing the sum of the squared differences between the target variable from the input dataset and the output of the independent variables' linear function (Wikipedia contributors, 2023b).

In this project, I was tasked with investigating factors determining engineering graduates' salaries, and building linear regression models for predictions by using the Engineering Graduate Salary Prediction dataset posted by KC (2020). Before going in-depth, here is a table summarizing everything I had achieved.

Priority	No.	Task	Status (%)
Required	1a	- Build a model from the first 11 features. - Compute MAE.	100
	1b	- Use k-fold cross-validation to find the best out of the 5 personality features. - Build a model from the best personality feature. - Compute MAE.	100
	1c	- Use k-fold cross-validation to find the best out of the 3 skill features. - Build a model from the best skill feature. - Compute MAE.	100
	1d	- Build at least 3 models. - Use k-fold cross-validation to find the best out of the new models. - Retrain the best model and compute MAE.	100
	2	- Document and discuss all progresses.	100

2 DATA, PACKAGES, AND FUNCTIONS

From this point onward,

- Every reference to `pandas` will be shortened to `pd`, `numpy` to `np`, `seaborn` to `sns`, `matplotlib.pyplot` to `plt`, `statsmodels.api` to `sm`, and cross-validation to CV.
- Functions with unspecified parameters will be written as `func(.)`.

2.1 Data Preparation

The original Engineering Graduate Salary Prediction dataset holding 2998 rows and 34 columns wouldn't be used in its entirety. Instead, Lecturer Phuong Uyen supplied the processed version which had had some problematic data filtered out.

- Columns with text values: `DOB`, `10board`, `12board`, `Specialization`, `CollegeState`.
- Columns with IDs and years: `ID`, `CollegeID`, `CollegeCityID`, `12graduation`, `GraduationYear`.

The new dataset still maintained 2998 rows but only 24 columns, with `Salary` designated as the target and the rest explanatory variables. Said data were then handed to us in the form of 2 files, i.e., `train.csv` and `test.csv`.

2.2 Package Usage

- *pandas* (Wes McKinney, 2010) and *NumPy* (Harris et al., 2020) were used to store, handle, and present data. All core calculations were also performed with their tools.
- *seaborn* (Waskom, 2021) and *Matplotlib*`.pyplot` (Hunter, 2007) were only for visualizing the correlation matrix with heat map.
- *statsmodels*`.api` (Seabold and Perktold, 2010) was used in **Stepwise Regression**, see Sect. 3.3.2, to procure features' p-values through fitting OLS models.

2.3 Function Usage

This section is indescribably redundant since the functions would be thoroughly explained in later sections anyway. I will only give a brief description for functions I deemed noteworthy. Starting with `lab04.ipynb`'s functions, provided by Lecturer Phuong Uyen.

- `OLSLinearRegression.fit(.)` fit, or trained, an OLS model.
- `OLSLinearRegression.get_params()` gave parameters estimated from the data, i.e., after training.
- `OLSLinearRegression.predict(.)` made predictions based on the testing data.
- `mae(.)` computed the **Mean Absolute Error** between the predictions and the target outcomes.

Self-coded functions.

- `kfold_traits(.)` performed k-fold CV on a training set to find the fittest feature, having the lowest average MAE.
- `stepwise_regress(.)` performed stepwise regression to remove insignificant features, whose p-values exceeded a certain limit.
- `corr_regress(.)` wielded the training set's correlation matrix to remove insignificant features. If two features shared a high correlation value, one will be discarded.
- `kfold_models(.)` performed k-fold CV on the given models to find the fittest one, having the lowest average MAE.
- `build_1dm1_stepReg(.)` built model `1dm1` with stepwise regression.
- `build_1dm2_corrMat(.)` built model `1dm2` with correlation matrix.
- `build_1dm3_featGroup(.)` built model `1dm3` by grouping features.
- `build_1dm4_corrStep(.)` built model `1dm4` with correlation matrix then stepwise regression.

Packages' functions.

- `pd.read_csv(.)` read `.csv` file into `pd.DataFrame`.
- `pd.DataFrame.sample(.)` returned a random sample of items from an axis of object.
- `pd.concat(.)` concatenated pandas objects along a particular axis.
- `pd.Series.to_frame(.)` converted Series to DataFrame.
- `pd.DataFrame.corr(.)` computed pairwise correlation of columns.
- `pd.DataFrame.drop(.)` dropped specified labels from rows or columns.
- `np.sum(.)`, `np.mean(.)`, `np.abs(.)`, `np.array(.)`, `np.full(.)`.
- `np.linalg.inv(.)` computed the multiplicative inverse of a matrix.
- `np.array_split(.)` split an array into multiple sub-arrays.
- `sm.OLS.fit(.)` did full fit of the model.
- `sns.heatmap(.)` plotted rectangular data as a color-encoded matrix.
- `plt.figure(.)`, `plt.title(.)`, `plt.show(.)`.

Python's base functions.

- `print(.)`, `range(.)`, `list.append(.)`, `zip(.)`, `all(.)`.
- `sorted(.)` returned a new list containing all items from the iterable in a specified order.

3 IDEAS AND IMPLEMENTATIONS

There are some concepts we need to preface, `train.csv` and `test.csv` were read into `train` and `test` `pd.DataFrame`s beforehand. Then, with `pd.DataFrame`'s `iloc` and `loc`, the following data could be derived hinging on situational needs.

- `train` fulfilled the role of fitting, or training, models. Yet, irrespective of them, it would always be split into `X_train`, a `pd.DataFrame` containing necessary independent variables, and `y_train`, `pd.Series` of the target's values.
- `test` was used sparingly and exclusively for appraising models. Similarly, `X_test` and `y_test` were obtained.

It is worth pointing out that I didn't employ `pd.DataFrame.to_numpy(.)`. So, extra care should be taken to ensure every `X` set was a `pd.DataFrame`, and every `y` set a `pd.Series`. Apart from k-fold CV, most of the time, operations were only done on columns, and the rows remained unaffected, thus, it was needless to generate new `y` sets.

3.1 1a. First 11 features

The features included: `Gender`, `10percentage`, `12percentage`, `CollegeTier`, `Degree`, `collegeGPA`, `CollegeCityTier`, `English`, `Logical`, `Quant`, `Domain`. The `X_1a_train` and `X_1a_test` sets were produced with `iloc[:, :11]`, other than that, the code was elementary.

3.2 1b, 1c. Five personality features, three skill features

3.2.1 Ideas

***K*-fold Cross-Validation** is a basic technique for evaluating predictive models. The dataset is divided into k subsets, or folds. Next, appointing a different fold as the validation set each time, the model is trained and evaluated k splits. Performance metrics from each fold are averaged reflecting the model's general performance (Pandian, 2023).

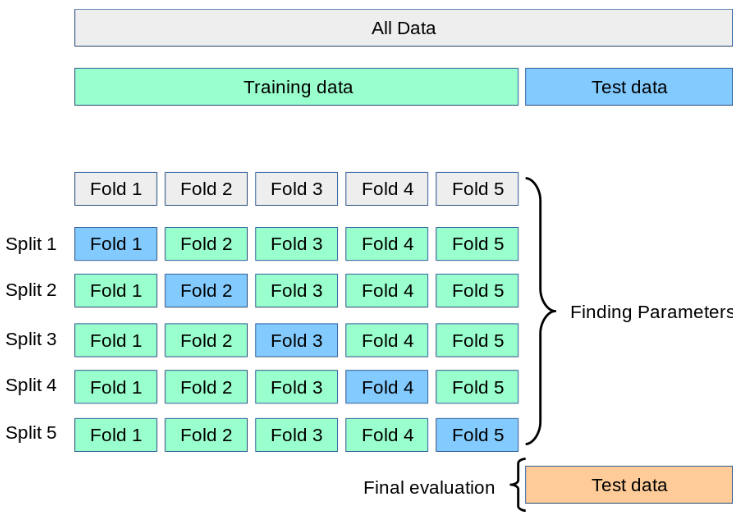

Figure 1. Visualization of a 5-fold CV process, courtesy of Pedregosa et al. (2023).

To grant fair judgements, the `train` set must be shuffled before the split. Furthermore, we have to guarantee that all models are assessed on the same `X` and `y` sets.

3.2.2 Implementations

```
def kfold_traits(train:pd.DataFrame, k_fold:int=5, seed:int=42,
↪ trait_label:str='Trait'):
    pass
```

The shuffling of `train` can be accomplished with `pd.DataFrame.sample(.)`. By setting `frac=1` and `replace=False`, the sampling would not take one row multiple times while keeping the sample's size to be identical to `train`'s. For more controlled outcomes, `random_state` should be set to the input `seed`. Then, with `np.array_split(train_shuffle, k_fold)`, the set is split into k equal parts (root and was on strike, 2022).

Entering the loop, in iteration i, fold i is used to form `X_subtest` and `y_subtest`, the remaining folds are merged through `pd.concat(.)` and that would be the source for `subtrain` sets. Now that all components have been formulated, for each feature, its related columns will be extracted to fit and rate a model, afterward, the resulting MAE is collected into a `list`. The `MAEs` `list` for every feature's model during this split will be added to a `split_MAEs` `list`, which compiles `MAEs` across all iterations.

After the loop, we convert `split_MAEs` into a `np.ndarray`, whose shape should be `(k_fold, feat_num)` representing an array of size k with each row a sub-array containing all MAEs of every feature's model during split i. To compute the average MAEs with respect to the features, we can use `np.mean(.)` with `axis=0`. Next, the array of average MAEs is cast back to `list` for labeling, by inserting sub-`list`s carrying a feature's name and its average MAE. Doing so would enable us to find the fittest feature as well via ascending-order sorting by average MAEs. In the end, we return the fittest feature as a `list` of its name plus average MAE, and the average MAEs collection as a `pd.DataFrame`.

With the crux of the task implemented, all that's left are examining the required features to set up the `train` sets.

- **Five personalities:** `conscientiousness`, `agreeableness`, `extraversion`, `nuerot↲icism`, `openess_to_experience`. As these happen to be the last features, we do `tra↲in.loc[:, 'conscientiousness':]` to also select the dependent variable `Salary`.
- **Three skills:** `English`, `Logical`, `Quant`. No snappy way around this, manually `trai↲n.loc[:, ['English', 'Logical', 'Quant', 'Salary']]` will have to suffice.

3.3 1dm1. Stepwise Regression

3.3.1 Ideas

Stepwise Regression is a process that assists in assuring the linear model produces the most precise predictions, by determining which factors are important. Certain variables have a higher p-value and were not meaningfully contributing to the model's accuracy.

- The detailed video titled Statistics 101: Model Building Methods - Forward, Backward, Stepwise, and Subsets by Foltz (2020) is a great guide for the concepts in this section.

Our stepwise regression will be carried out with a backwards elimination approach. Initially, all variables are involved, and in each step, the most inconsequential variable is dropped. This is repeated until all variables left over are statistically significant (Kwok, 2021).

3.3.2 Implementations

```
def stepwise_regress(X_train:pd.DataFrame, y_train:pd.Series, pval_max:float=0.05):
    pass
```

The features' p-values can be gathered by drawing on `statsmodels.api` library. A model is fit to the columns of interest giving us an object, named `stats`, of the `statsmodels.regression.linear_model.OLSResults` class. Then, calling the `pvalues` property on `stats`, the p-value of every variable is obtained.

With the p-values in hand, the rest turns trivial. `all(stats.pvalues <= pval_max)` checks whether all p-values stay within the limit. If this was not the case, we would continue seeking out the most statistically insignificant feature, i.e., having the highest p-value, and removing its column with `pandas.DataFrame.drop(.)`. For each variable-dropping instance, the model is refitted to the modified `train` set. When our `all(.)` condition is satisfied, the loop is stopped and all enduring features' names are returned through `stats.model.exog_names` (user1074057, 2017), for completeness' sake, the target variable is also thrown in.

3.4 1dm2. Correlation Matrix

3.4.1 Ideas

Correlation is a statistical term commonly refers to how close two variables are to having a linear relationship with each other. Features with high correlation are more linearly dependent resulting in them sharing almost the same influence on the dependent variable. Hence, when a high-correlation couple exists, it's safe to drop one of the two features (R, 2018).

In practice, the linear dependence between pairs of features is measured into the **Correlation Matrix** – a square matrix that contains the Pearson Product-Moment Correlation coefficients. Such coefficients are in the $[-1, 1]$ range. Two features have a perfect positive correlation if $r = 1$, no correlation if $r = 0$, and a perfect negative correlation if $r = -1$ (Pragati, 2023).

3.4.2 Implementations

```
def corr_regress(train:pd.DataFrame, corr_max:float=0.9):
    pass
```

Assembling a correlation matrix out of `train` is as simple as invoking `pandas.DataFrame.corr(.)`. To mark which variable will be kept, we can use *NumPy*'s masking technique, where **`True`** means "keep" and **`False`** "discard". First, we initialize a `columns` array with every feature tagged as **`True`** through `np.full(.)`. Next, we loop through each correlation `ij`, if its value surpasses `corr_max`, feature `j` is set to **`False`** in `columns`. Finally, due to `columns` comprising of only `booleans`, it's essentially our mask. Passing it to the columns of `train` and taking their values, we end up with the names of features that are non-linearly dependent. Just like in Sect. 3.3.2, the target variable and correlation matrix should also be returned.

3.5 1dm3, 1dm4. Feature Grouping, Correlation & Stepwise

1dm3 Feature Grouping is much more tedious than it is complex. There was absolutely no scientific basis backing this method, I merely drew inspiration from the other tasks and spontaneously made it. Basically, a bunch of `pd.DataFrame.locs` and `np.sum(.)`s are involved for merging columns, then `pd.concat(.)` to establish a new `pd.DataFrame`. We only need to pay attention to working on `axis=1`, which is the "horizontal" plane, i.e., the axis of columns.

1dm4 Correlation & Stepwise is just a combination of correlation matrix and stepwise regression, as suggested in Feature selection — Correlation and P-value by R (2018). Our stepwise regression, explored in Sect. 3.3, did selection based on p-values so it is practically an identical routine to the article's. First, we use `corr_regress(.)`, Sect. 3.4.2, to preserve solely non-linearly dependent variables, then a pass through `stepwise_regress(.)`, Sect. 3.3.2, to further filter out statistically insignificant features.

3.6 1d. Model Comparison

Once again, **K-fold Cross-Validation**, Sect. 3.2.1, is recruited for appraisal. However, unlike `kfold_traits(.)` from Sect. 3.2.2, on top of a single `train` set, we have to slide in multiple different models. From this, a call for neat implementations that avoid both code duplication while also maintain portability arises.

The system I proposed is to construct `model_builders`, a collection of functions possessing matching parameters and return patterns. Going forward, rating our models can then be reframed to gauging said builders. A generic `build_model(.)` will take in `X_train`, `y_train`, `X_test`, perform its respective variable manipulation and subsequently return the freshly fitted model along with the altered `X_test`.

```
def kfold_models(model_builders:list, train:pd.DataFrame, k_fold:int=5, seed:int=42):
    pass
```

`kfold_models(.)` abides by exactly the very logic of `kfold_traits(.)` from Sect. 3.2.2. The only changes are the replacement of `traits` to `model_builders`, and making sure that the best model's builder is incorporated as a returnee. Another point of note, because the model was abstracted away, to know which features persist until the final equation, after the `OLSLinearRegression.get_params()` method yielding us a `pd.Series`, we can utilize a conjunction of `pd.Series.index = pd.DataFrame.columns.to_numpy(.)` which associates the model's parameters to their appropriate variables' name (Tavory, 2015).

4 RESULTS AND DISCUSSION

All k-fold CVs were done with $k = 5$, and every shuffling with `pd.DataFrame.sample(.)` had its `random_state` set to 42 for reproducibility (Sahagian, 2020).

4.1 1a. First 11 features

$$\begin{aligned}\text{Salary} = &-22756.513 \times \text{Gender} + 804.503 \times \text{10percentage} + 1294.655 \times \text{12percentage} \\ &- 91781.898 \times \text{CollegeTier} + 23182.389 \times \text{Degree} + 1437.549 \times \text{collegeGPA} \\ &- 8570.662 \times \text{CollegeCityTier} + 147.858 \times \text{English} + 152.888 \times \text{Logical} \\ &+ 117.222 \times \text{Quant} + 34552.286 \times \text{Domain} \\ \text{MAE} = &\ 104863.778\end{aligned} \tag{1}$$

4.2 1b, 1c. Five personality features, three skill features

$$\begin{aligned} \text{Salary} &= -56546.304 \times \text{nueroticism} \\ \text{MAE} &= 291019.693 \end{aligned} \tag{2}$$

Table 1. Results of `kfold_traits(train_1b)`.

No.	Personality	Average MAE
0	conscientiousness	306309.202
1	agreeableness	300912.678
2	extraversion	307030.103
3	nueroticism	299590.050
4	openess_to_experience	302957.692

$$\begin{aligned} \text{Salary} &= 585.895 \times \text{Quant} \\ \text{MAE} &= 106819.578 \end{aligned} \tag{3}$$

Table 2. Results of `kfold_traits(train_1c)`.

No.	Skill	Average MAE
0	English	121925.884
1	Logical	120274.778
2	Quant	118124.524

It can clearly be observed in Table 1 and 2 that the features with the lowest average MAE after performing k-fold CVs were chosen to be the main ingredients of the models, see Eq. 2 and 3.

4.3 1d. Model Comparison

`stepwise_regress`'s `pval_max` was set to 0.05 (Bruin, 2011), and `corr_regress`'s `corr_max` 0.9 (R, 2018).

Table 3. Results of `kfold_models(model_builders, train)`.

No.	Model	Average MAE
0	1dm1 Stepwise Regression	110684.300
1	1dm2 Correlation Matrix	110420.414
2	1dm3 Feature Grouping	115207.564
3	1dm4 Correlation & Stepwise	110684.300

This section finally allows us some materials for discussion. First, **1dm2** is actually a full-feature model. The reason for this can be inferred from Fig. 2, we didn't have any features' pair being too linearly dependent on each other, i.e., their correlation exceeded 0.9, so `corr_regress(.)` didn't filter out a single variable. This also explains why **1dm1** and **1dm4** shared the same number, because correlation refused to do anything, it's identical to passing the raw set to `stepwise_regress(.)`. In that sense, we were realistically dealing with just 3 models.

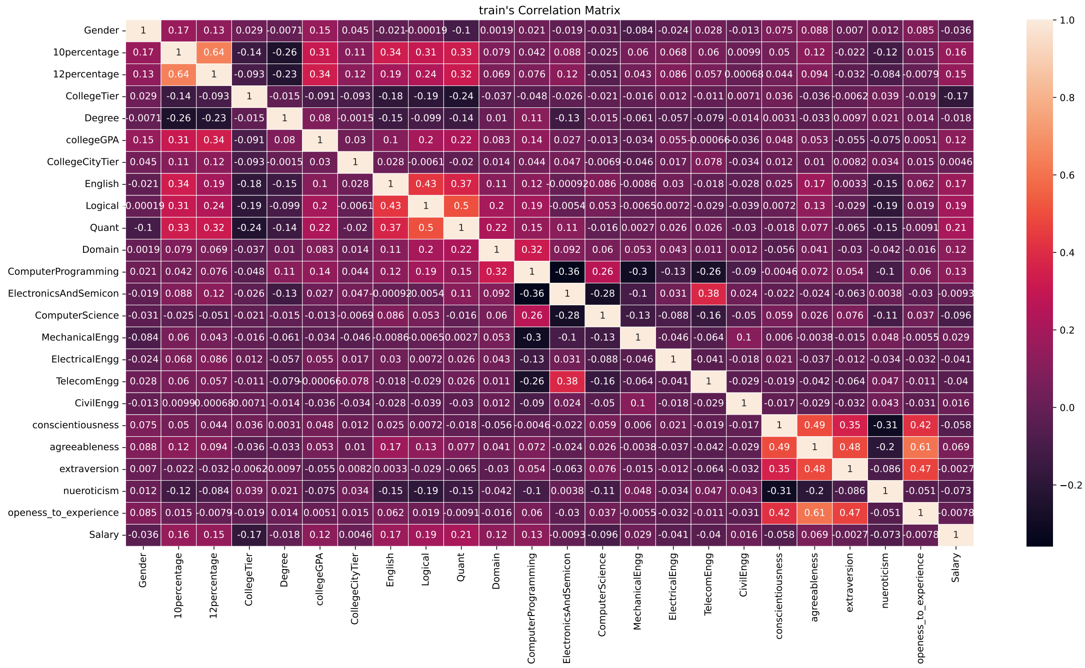

Figure 2. Correlation matrix of `train` set.

A small tidbit, for the original `train` and `test` sets, **1md1** surprisingly produced lower MAE, still, it lost to **1dm2** when we considered things averagely. This will be elaborated on in App. A. Anyway, here is the **1dm2** model.

$$\begin{aligned}
\text{Salary} = {} & -23874.542 \times \text{Gender} + 898.576 \times \text{10percentage} + 1203.496 \times \text{12percentage} \\
& - 83592.388 \times \text{CollegeTier} + 11515.431 \times \text{Degree} + 1626.519 \times \text{collegeGPA} \\
& - 5717.734 \times \text{CollegeCityTier} + 153.435 \times \text{English} + 120.511 \times \text{Logical} \\
& + 102.581 \times \text{Quant} + 27939.640 \times \text{Domain} + 76.730 \times \text{ComputerProgramming} \\
& - 47.747 \times \text{ElectronicsAndSemicon} - 177.388 \times \text{ComputerScience} \\
& + 33.933 \times \text{MechanicalEngg} - 151.471 \times \text{ElectricalEngg} - 64.198 \times \text{TelecomEngg} \\
& + 145.895 \times \text{CivilEngg} - 19814.830 \times \text{conscientiousness} + 15503.267 \times \text{agreeableness} \\
& + 4908.582 \times \text{extraversion} - 10661.029 \times \text{nueroticism} \\
& - 5815.021 \times \text{openess_to_experience} \\
\text{MAE} = {} & 101872.211
\end{aligned} \tag{4}$$

SOFTWARE CITATIONS

This work uses the following software and packages:

- Python 3.11.4 (Van Rossum and Drake, 2009)
- NumPy 1.25.0 (Harris et al., 2020)
- pandas 2.0.3 (Wes McKinney, 2010)
- seaborn 0.12.2 (Waskom, 2021)

- Matplotlib 3.7.1 (Hunter, 2007)
- statsmodels 0.14.0 (Seabold and Perktold, 2010)

DATA AVAILABILITY

All data and software used in this paper are public, their links are provided in the text when discussed. All data were used for educational and research purposes.

ACKNOWLEDGEMENTS

I would like thank my lecturers of the current Applied Mathematics and Statistics course at Ho Chi Minh City University of Science (Vu Quoc Hoang, Nguyen Van Quang Huy, Le Thanh Tung, Phan Thi Phuong Uyen) for their guidance and support during theory classes and lab sessions. Overall, this project was quite insightful, which could have been better if it wasn't for the existence of a single requirement being absolutely a drag to get through, listing all functions used, refer to Sect. 2.3. In my honest opinion, that task was exasperating.

REFERENCES

J. Brownlee. How to transform target variables for regression in python. *Machine Learning Mastery*, 2020. URL https://machinelearningmastery.com/how-to-transform-target-variables-for-regression-with-scikit-learn/. [Accessed 14-August-2023].

J. Bruin. Regression analysis — stata annotated output. *UCLA Office of Advanced Research Computing's Statistical Methods and Data Analytics*, 2011. URL https://stats.oarc.ucla.edu/stata/output/regression-analysis/. [Accessed 14-August-2023].

B. Foltz. Statistics 101: Model building methods - forward, backward, stepwise, and subsets. YouTube, 2020. URL https://youtu.be/-inJu1jHqb8. [Accessed 12-August-2023].

B. Foltz. Statistics 101: Variable transformations, improving a model. YouTube, 2021. URL https://youtu.be/fyRubPKgVoY. [Accessed 14-August-2023].

C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9 (3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

IrishStat. When (and why) should you take the log of a distribution (of numbers)? Cross Validated, 2021. URL https://stats.stackexchange.com/q/18852. (version: 2021-12-22).

M. KC. Engineering graduate salary prediction, 2020. URL https://www.kaggle.com/datasets/manishkc06/engineering-graduate-salary-prediction. [Retrieved 09-August-2023].

R. Kwok. Stepwise regression tutorial in python. *Towards Data Science*, 2021. URL https://towardsdatascience.com/stepwise-regression-tutorial-in-python-ebf7c782c922. [Accessed 12-August-2023].

S. Pandian. K-fold cross validation technique and its essentials. *Analytics Vidhya*, 2023. URL https://www.analyticsvidhya.com/blog/2022/02/k-fold-cross-validation-technique-and-its-essentials/. [Accessed 12-August-2023].

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 3.1. cross-validation: evaluating estimator performance. *scikit-learn's User Guide*, 2023. URL https://scikit-learn.org/stable/modules/cross_validation.html.

Pragati. Find correlation between features and target using the correlation matrix. *Knowledge Transfer's For Machine Learning*, 2023. URL https://androidkt.com/find-correlation-between-features-and-target-using-the-correlation-matrix/. [Accessed 13-August-2023].

V. R. Feature selection — correlation and p-value. *Machine Learning - The Science, The Engineering, and The Ops*, 2018. URL https://vishalramesh.substack.com/p/feature-selection-correlation-and-p-value-da8921bfb3cf?s=w. [Accessed 13-August-2023].

root and C. was on strike. Split a large pandas dataframe. Stack Overflow, 2022. URL https://stackoverflow.com/a/17315875. (version: 26-Jun-2022).

G. Sahagian. What is random state? and why is it always 42? *Medium*, 2020. URL https://grsahagian.medium.com/what-is-random-state-42-d803402ee76b. [Accessed 14-August-2023].

S. Seabold and J. Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

A. Tavory. Pandas: Printing the names and values in a series. Stack Overflow, 2015. URL https://stackoverflow.com/a/30523731. [Accessed 13-August-2023].

user1074057. Pulling variable names when using pandas and statsmodels. Stack Overflow, 2017. URL https://stackoverflow.com/q/11836286. [Accessed 12-August-2023].

G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

M. L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60): 3021, 2021. doi: 10.21105/joss.03021. URL https://doi.org/10.21105/joss.03021.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

Wikipedia contributors. Linear regression — Wikipedia, the free encyclopedia, 2023a. URL https://en.wikipedia.org/w/index.php?title=Linear_regression&oldid=1168216625. [Online; accessed 11-August-2023].

Wikipedia contributors. Ordinary least squares — Wikipedia, the free encyclopedia, 2023b. URL https://en.wikipedia.org/w/index.php?title=Ordinary_least_squares&oldid=1164085874. [Online; accessed 11-August-2023].

A 1d. FITTING AND TESTING LEFT-OVER MODELS

All models were fit and assessed on the original `train` and `test` sets.

$$\begin{aligned}\text{Salary} = &-23848.381 \times \text{Gender} + 1655.743 \times \text{12percentage} - 77100.869 \times \text{CollegeTier}\\ &+ 1924.288 \times \text{collegeGPA} + 170.549 \times \text{English} + 134.509 \times \text{Logical}\\ &+ 105.165 \times \text{Quant} + 29963.516 \times \text{Domain} + 68.962 \times \text{ComputerProgramming}\\ &- 72.831 \times \text{ElectronicsAndSemicon} - 179.128 \times \text{ComputerScience}\\ &- 154.514 \times \text{ElectricalEngg} - 20149.823 \times \text{conscientiousness}\\ &+ 14170.689 \times \text{agreeableness} - 11280.629 \times \text{nueroticism}\\ \text{MAE} = &\ 101775.812\end{aligned} \tag{5}$$

1dm1's equation is much more appealing when compared to the bulky Eq. 4 of **1dm2**. On top of that, there was also a noticeable improvement in MAE. We won't be talking about **1dm4** since it basically is **1dm1**, as mentioned in Sect. 4.3.

$$\begin{aligned}\text{Salary} = &\ 355.425 \times \text{Gen2Col} + 168.107 \times \text{Eng2Qua} - 43.475 \times \text{Dom2Civ}\\ &- 1611.069 \times \text{con2ope}\\ \text{MAE} = &\ 108537.032\end{aligned} \tag{6}$$

As for **1dm3**, albeit a highly elegant equation, the MAE was expectedly unacceptable.

B CONCEPTS THAT I FAILED TO IMPLEMENT

Another technique I did come across regarding model building is **Variable Transformation**.

- Statistics 101: Variable Transformations, Improving a Model by Foltz (2021).
- When (and why) should you take the log of a distribution (of numbers)? by IrishStat (2021).
- How to Transform Target Variables for Regression in Python by Brownlee (2020).

Unfortunately, all the stuffs about plots and R^2 analysis went way over my head, and I felt like I didn't have enough experience to correctly judge when certain transformations were required. Diving into something blindly might lead to substandard performance, or even worse, erroneous results, so I ultimately gave up.