# Regression with Regularization

Mentor: Pararawendy Indarjo

Hey I'm,

# Pararawendy Indarjo

I am a,

- CURRENTLY | **Senior DS at Bukalapak**
- 19 - 20 | **Data Analyst at Eureka.ai**

Universiteit Leiden

BSc Mathematics

MSc Mathematics

Linkedin :
https://www.linkedin.com/in/pararawendy-indarjo/
Blog : medium.com/@pararawendy19

# Outline

- Bias-Variance Trade-Off in ML Models
- Regularization in Linear Regression
- Ridge Regression
- LASSO
- Choosing the best regularization parameter
- Assignment

# Bias & Variance in ML Models

01

## Bias

- Model **systematically deviates** from the true data pattern
- i.e. discrepancy between average model  prediction and ground truth
- **High bias** is usually associated with **underfitting**

02

## Variance

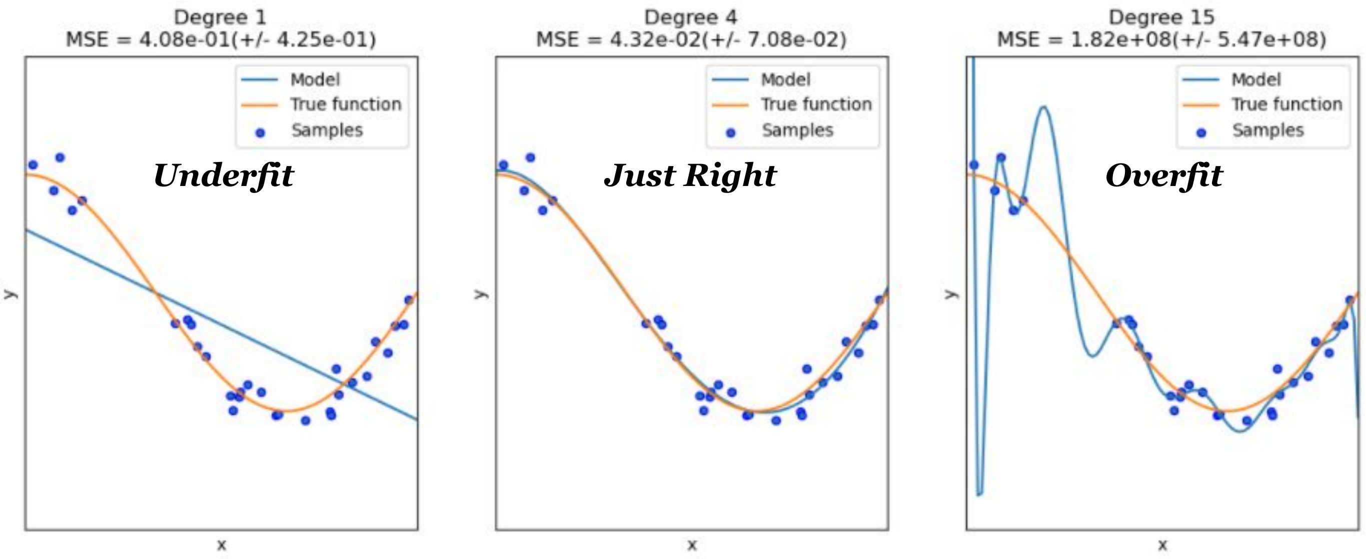
- **Variability in the model prediction**
- i.e. small change on the data, leads to significant change on the model output
- **High variance** is usually associated with **overfitting**

# Underfitting

- Overly-simplified model
- Indication: high error (evaluation metrics) on both train and test data

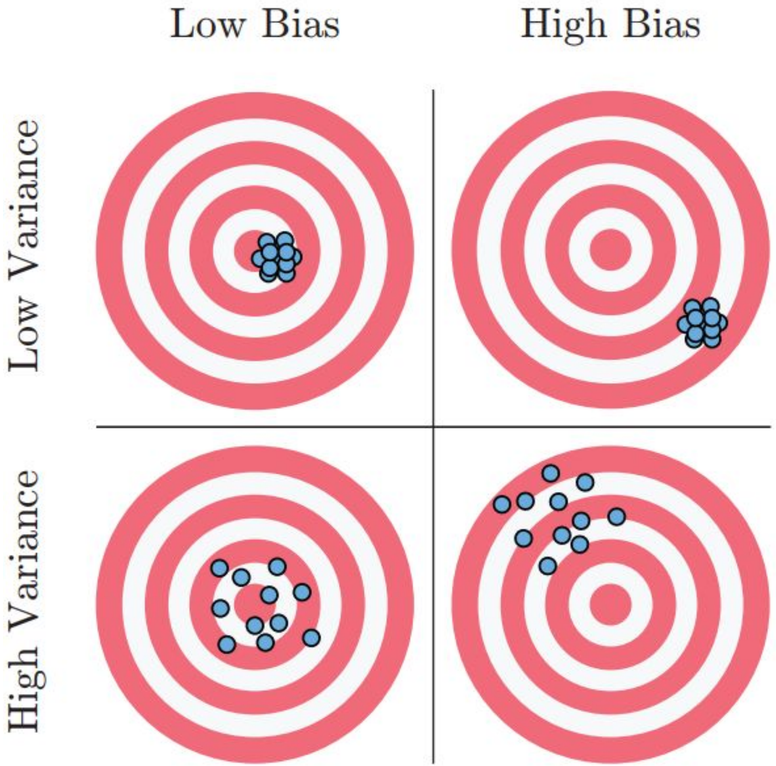
# Overfitting

- Overly-complex model
- Start modeling the noise, instead of the true pattern of the data
- Indication: **low** training error, **high** test error



# Bias & Variance Illustration

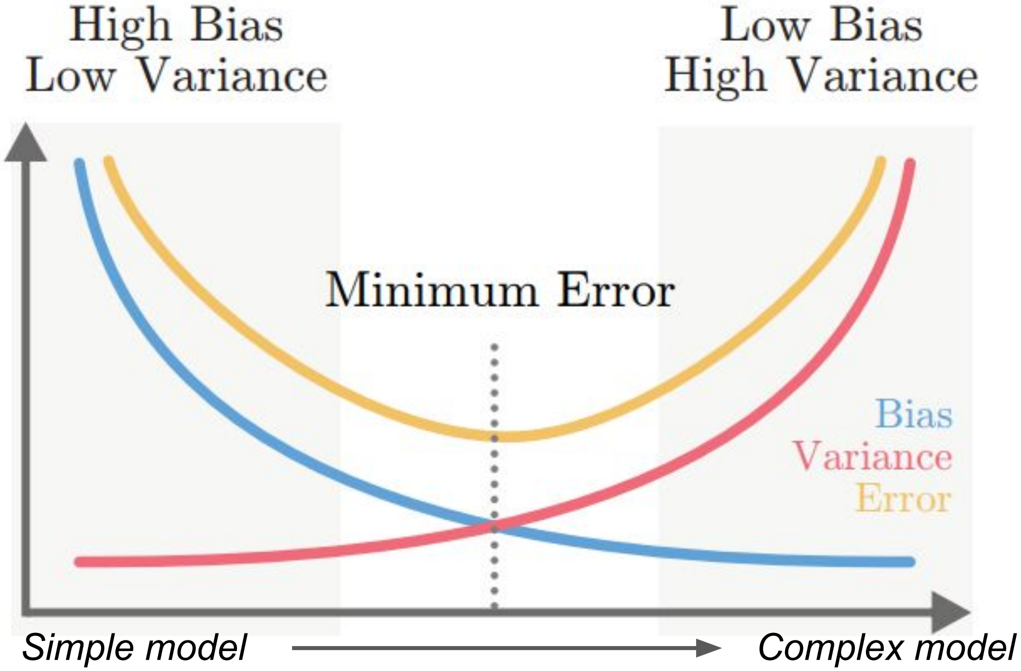
Remember the concepts via dart board pictures



# Bias-Variance Trade-Off

We want the sweet-spot between the two

- 3 sources of model error
  - Bias
  - Variance
  - Irreducible error (inherent noise)



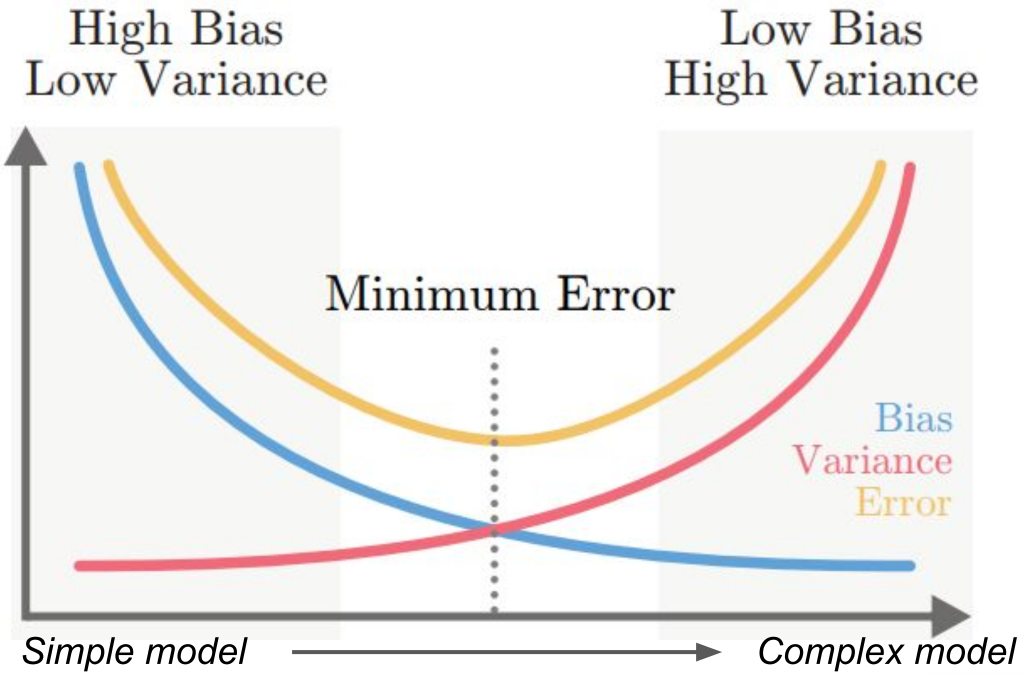
# Bias-Variance Trade-Off

We want the sweet-spot between the two

- 3 sources of model error
  - Bias
  - Variance
  - Irreducible error (inherent noise)

- Therefore, we can
  - reduce bias by increasing variance
    - E.g. use more complex model
  - reduce variance by increasing bias
    - E.g. use simpler model, including **model with regularization**

- The ideal state is to find the **balance between bias and variance** → minimum model error



# Regularization in Linear Regression

- Regularization (penalization):
  - Make the model more regular/simple
  - By shrinking the model coefficients towards zero
    - I.e. coefficients with smaller absolute values

- Objective:
  - To reduce model variance, by (slightly) increasing the model bias
  - Ultimately, **to address overfitting**

# Objective/Loss Function

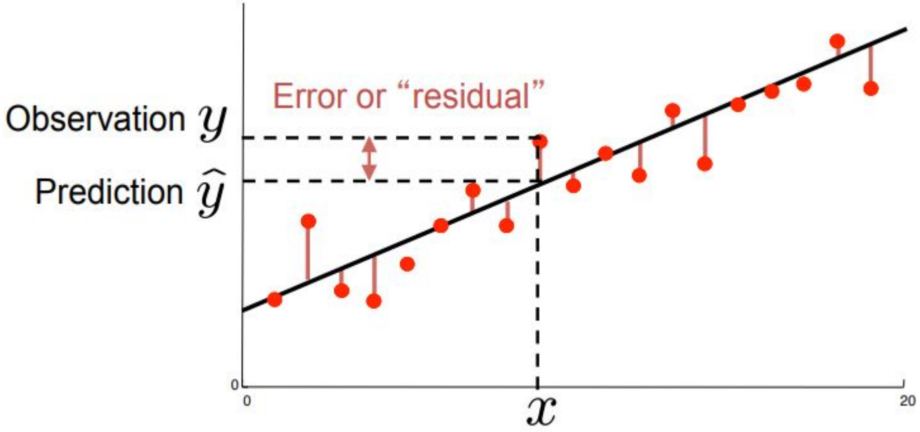
*RECAP*

The magnitude we want to minimize when training the model

- **Residual**: delta between the real target variable *y* and the predicted target variable $y_{hat}$
  - Predicted values = regression line

- Train the model to minimize: **Residual sum of squares**

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2$$




- **red points** are our data points
- **black line** is the regression line (our prediction values)

# Ridge

- Linear regression with modified loss function

$$\sum_i (y_i - \hat{y}_i)^2 + \lambda \sum_j \boxed{\beta_j^2}$$

- Where **lambda is non-negative** regularization parameter
  - Larger == heavier regularization
  - 0 == ordinary linear regression

- Effect to the model coefficients (beta)
  - Beta will become smaller (larger lambda == smaller beta)
  - **Yet they all <u>never</u> be exactly zero**

# LASSO

- Linear regression with modified loss function

$$\sum_i (y_i - \hat{y}_i)^2 + \lambda \sum_j \boxed{|\beta_j|}$$

- Where **lambda is non-negative** regularization parameter
  - Larger == heavier regularization
  - 0 == ordinary linear regression

- Effect to the model coefficients (beta)
  - Beta will become smaller (larger lambda == smaller beta)
  - **Some of them will be <u>exactly zero</u> (eliminated from the model)**

*LASSO:* Least Absolute Shrinkage and Selection Operator

Any

# Question

# Modeling Flow: Regularized Regression

1. **Split the data**
   - 80 : 20 is fine

2. **Multicollinearity check**
   - Calculate VIF score for each feature
   - Correlation analysis to drop redundant feature

3. **Fit the model on training data**
   - Define and fit Lasso() or Ridge()
   - Only include retained features from step 2

4. **Model diagnostic**
   - Residual plot
   - R2 score on training data
   - *No need QQ-plot*
     - *Not assuming normally distributed residuals*

5. **Evaluate the model on test data**
   - RMSE
   - MAE ← *New for today*
   - MAPE ← *New for today*

# The Data Used

- For the rest of the lecture, we will
  `regression_data.csv`

- The data is about university admit probability, based on several applicant’s features
  - GRE score
  - TOEFL score
  - University ranking
  - Motivation letter quality
  - Recommendation letter strength
  - GPA
  - Research experience

# Split the Data

Using train_test_split() function

- Recall, we want to predict `admit_prob`

```
# split train test
from sklearn.model_selection import train_test_split

feature = admit.drop(columns='admit_prob')
target = admit[['admit_prob']]

ftr_train, ftr_test, tgt_train, tgt_test = train_test_split(feature,
                                                            target,
                                                            test_size=0.20,
                                                            random_state=42)
```

# Multicollinearity (1/2)

Variance Inflation Factor

- Multicollinearity: when two or more features/predictors are **highly correlated** each other

- This can cause coefficients of estimates become unreliable
    - i.e.  little change in the training data leads to completely different learned coefficients

- We can detect this by computing **Variance Inflation Factor (VIF)** for each feature

$$\mathrm{VIF}(\hat{\beta}_j) = \frac{1}{1 - R^2_{X_j | X_{-j}}}$$

    - On a high level: **VIF feature j will be high if j can be predicted using the rest of other features**. Vice versa
    - VIF == 1 → No multicollinearity
    - VIF between 4 and 10 → Moderate multicollinearity
    - VIF > 10 →  Severe multicollinearity

# Multicollinearity (2/2)

Using statsmodels library

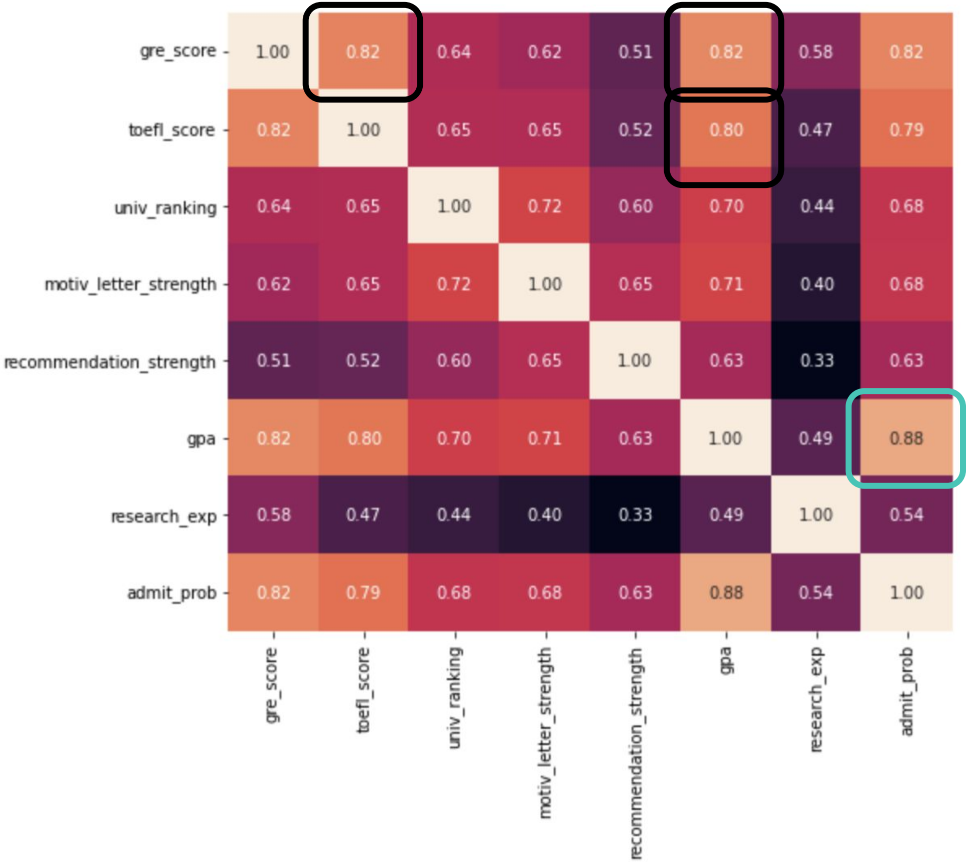
```
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
from statsmodels.tools.tools import add_constant

X = add_constant(feature_admit_train)

vif_df = pd.DataFrame([vif(X.values, i)
               for i in range(X.shape[1])],
              index=X.columns).reset_index()
vif_df.columns = ['feature','vif_score']
vif_df = vif_df.loc[vif_df.feature!='const']
```

| | feature | vif_score |
|---|---|---|
| 1 | gre_score | 4.489983 |
| 2 | toefl_score | 3.664298 |
| 3 | univ_ranking | 2.572110 |
| 4 | motiv_letter_strength | 2.785764 |
| 5 | recommendation_strength | 1.977698 |
| 6 | gpa | 4.654540 |
| 7 | research_exp | 1.518065 |

# Feature Correlation

To prevent multicollinearity

- We can draw a correlation heatmap
    - Using `sns.heatmap()`
- We found that `gre_score`, `toefl_score`, and `gpa` are highly correlated each other
    - We decide to **include only `gpa`** to represent these three features
    - Because it is the most correlated with the target variable
- ***Note: Threshold: abs(corr) >= 0.8***



# Training the Model

Excluding `gre_score` and `toefl_score`

```
feature_admit_train = feature_admit_train.drop(columns=['gre_score','toefl_score'])
feature_admit_test = feature_admit_test.drop(columns=['gre_score','toefl_score'])
```

```
from sklearn.linear_model import Ridge

# train the model
X_admit_train = feature_admit_train.to_numpy()
y_admit_train = target_admit_train.to_numpy()

# define the model
ridge_reg = Ridge(alpha=0.1,
                  random_state=42)

ridge_reg.fit(X_admit_train, y_admit_train)
```

*This one is **lambda***

For lasso, only need to change `Ridge` to `Lasso`

# Comparing the Coefficients

**Ridge**

| | feature | coefficient |
|---|---|---|
| 0 | intercept | -0.764483 |
| 1 | univ_ranking | 0.007031 |
| 2 | motiv_letter_strength | 0.004406 |
| 3 | recommendation_strength | 0.014806 |
| 4 | gpa | 0.160723 |
| 5 | research_exp | 0.038290 |

**Lasso**

| | feature | coefficient |
|---|---|---|
| 0 | intercept | 0.702539 |
| 1 | univ_ranking | 0.006951 |
| 2 | motiv_letter_strength | 0.000000 |
| 3 | recommendation_strength | 0.000000 |
| 4 | gpa | 0.000000 |
| 5 | research_exp | 0.000000 |

*exactly zero*

# Hands-On

- Open today's Jupyter notebook on your Google Colab!
- Make sure you have uploaded the required CSV files to your google drive
    - Remember the file path!

# Comparing the Coefficients

**Ridge**

| | feature | coefficient |
|---|---|---|
| **0** | intercept | -0.764483 |
| **1** | univ_ranking | 0.007031 |
| **2** | motiv_letter_strength | 0.004406 |
| **3** | recommendation_strength | 0.014806 |
| **4** | gpa | 0.160723 |
| **5** | research_exp | 0.038290 |

**Lasso**

| | feature | coefficient |
|---|---|---|
| **0** | intercept | 0.702539 |
| **1** | univ_ranking | 0.006951 |
| **2** | motiv_letter_strength | 0.000000 |
| **3** | recommendation_strength | 0.000000 |
| **4** | gpa | 0.000000 |
| **5** | research_exp | 0.000000 |

*exactly zero*

***But…***
***How to select the best lambda?***

# Modeling Flow: Regularized Regression (with choosing lambda)

1. **Split data: train - validate - test**
   - 80% → 80% training vs 20% validation
   - 20% testing
2. **Multicollinearity check**
   - Calculate VIF score for each feature
   - Correlation analysis to drop redundant feature
3. **Fit multiple models on training data**
   - Train multiple models with different lambdas (alpha)
   - Only include retained features from step 2
4. **Choose the best lambda from validation set**
   - The one with the smallest RMSE
5. **Model diagnostic**
   - Residual plot, R2 score on training data
6. **Evaluate the model on test data**
   - RMSE
   - MAE ← *New for today*
   - MAPE ← *New for today*

# Split the Data

We now have three data: train-validation-test

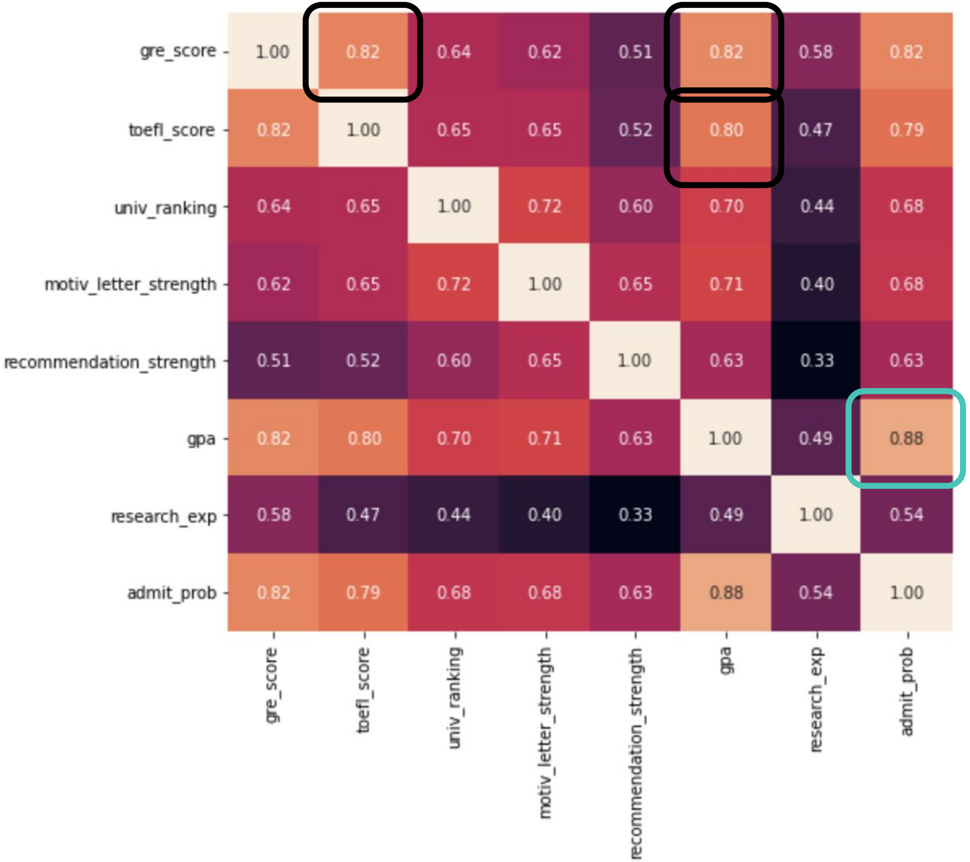
- First, split the data into **pre-train** and **testing** (80:20)
  - Using `train_test_split(feature, target)`

- Next, split pre-train data into **train** and **validation** (80:20)
  - Using `train_test_split(feature_pretrain, target_pretrain)`

# Feature Correlation

To prevent multicollinearity

- Same as previously, will drop:
    - `gre_score`
    - `toefl_score`



# Train Multiple Models

Lambda (alpha) in [0.01, 0.1, 1, 10]

```
from sklearn.linear_model import Ridge

# train the model
X_admit_train = feature_admit_train.to_numpy()
y_admit_train = target_admit_train.to_numpy()

# define the model
ridge_reg_pointzeroone = Ridge(alpha=0.01, random_state=42)
ridge_reg_pointone = Ridge(alpha=0.1, random_state=42)
ridge_reg_one = Ridge(alpha=1, random_state=42)
ridge_reg_ten = Ridge(alpha=10, random_state=42)

# fit the model (training)
ridge_reg_pointzeroone.fit(X_admit_train, y_admit_train)
ridge_reg_pointone.fit(X_admit_train, y_admit_train)
ridge_reg_one.fit(X_admit_train, y_admit_train)
ridge_reg_ten.fit(X_admit_train, y_admit_train)
```

# Choosing the Best Lambda

Using validation data

```
alphas = [0.01, 0.1, 1., 10]
models = [ridge_reg_pointzeroone,
          ridge_reg_pointone,
          ridge_reg_one,
          ridge_reg_ten]

for model, alpha in zip(models, alphas):
    y_predict_validation = model.predict(X_admit_validation)
    rmse = np.sqrt(mean_squared_error(y_admit_validation,y_predict_validation))
    print(f'RMSE of Ridge regression model with alpha = {alpha} is {rmse}')
```

```
RMSE of Ridge regression model with alpha = 0.01 is 0.05732468266149709
RMSE of Ridge regression model with alpha = 0.1 is 0.0573423796471404
RMSE of Ridge regression model with alpha = 1.0 is 0.05752869822295276
RMSE of Ridge regression model with alpha = 10 is 0.0598318401121286З
```

*The best lambda is 0.01*

# Interpreting the Best Model

Best model == model with the best lambda

| | feature | coefficient |
|---|---|---|
| 0 | intercept | -0.741404 |
| 1 | univ_ranking | 0.005556 |
| 2 | motiv_letter_strength | 0.009497 |
| 3 | recommendation_strength | 0.015778 |
| 4 | gpa | 0.155693 |
| 5 | research_exp | 0.042721 |

```
admit_prob = -0.741 +
     0.005 uni_rank + 0.009 motiv +
     0.015 recom + 0.155 gpa +
     0.042 research
```

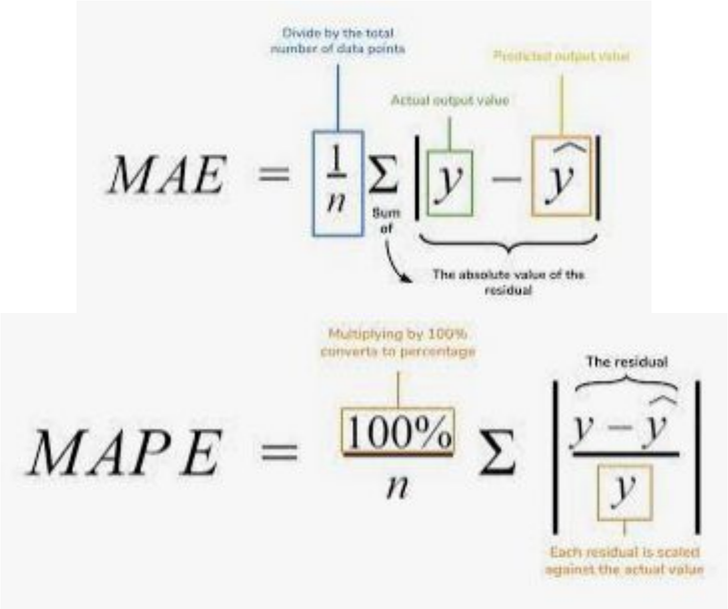
Interpretation?

# Hands-On

- Open today's Jupyter notebook on your Google Colab!
- Make sure you have uploaded the required CSV files to your google drive
  - Remember the file path!

# Model Evaluation (1/2)

- After we are satisfied (enough) with the model, we perform model evaluation
- Essentially, we check how is the **model performance on test data**
- To do so, we will compute these two regression metrics
  - Mean absolute error (MAE):
    - How far **in absolute basis** is the model's prediction from the actual data on average
  - Mean absolute percentage error (MAPE)
    - How far **in relative percentage basis** is the model's prediction **relative to the actual data** on average



# Model Evaluation (2/2)

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error

mean_absolute_error(y_admit_test, y_predict_test)
mean_absolute_percentage_error(y_admit_test, y_predict_test)
```

```
MAE for testing data is 0.039665519822843456
MAPE for testing data is 0.0646079497445827
```

- MAE = 0.040
  - On average, our prediction deviates the true `admit_prob` by 0.040
- MAPE = 0.065 = 6.5%
  - Moreover, this 0.040 is equivalent to 6.5% deviation relative to the true `admit_prob`

# When to Use: Ridge vs LASSO

01

## Ridge

- Only make the coefficients small, NOT zero
- Works well if there are many large parameters of about the same value

02

## LASSO

- Can set some coefficients to zero
  - Thus performing variable selection
- Tends to do well if there are a small number of significant parameters and the others are close to zero

***That said, ridge and LASSO oftentimes perform similarly, just train both and choose the best!***

# Any
# Question

# Hands-On

- Open today's Jupyter notebook on your Google Colab!
- Make sure you have uploaded the required CSV files to your google drive
  - Remember the file path!

# Assignment

- What to submit? Google colab link (don’t forget to share access to me:  pararawendy19@gmail.com)
    - Format notebook name: HW_REGRESSION_<YOUR COMPLETE NAME>

# The Data Used

- For the assignment, we will use the following data
  - [Download link]
- The data is about predicting housing price (`medv`) in Boston city, features:
  - Criminal rate (crim)
  - Residential land zoned proportion (zn)
  - Non-retail business acres proportion (indus)
  - Is bounds with river (chas)
  - Nitrogen oxides concentration (nox)
  - Number rooms average (rm)
  - Owner age proportion (age)
  - Weighted distance to cities (dis)
  - Accessibility index (rad)
  - Tax rate (tax)
  - Pupil-teacher ratio (ptratio)
  - Black proportion (black)
  - Percent lower status (lstat)

# Assignment

## Instructions

1. Split data: train - validate - test *(point: 10)*
2. Draw correlation plot on training data and perform feature selection on highly correlated features *(point: 10)*
3. Fit models on training data (lambdas = [0.01, 0.1, 1, 10]) *(point:50)*
    a. Ridge regression *(point: 25)*
    b. LASSO *(point: 25)*
4. Choose the best lambda from the validation set *(point: 20)*
    a. Use RMSE as metric
    b. Interpret a sample of the coefficients of the best model
        i. Ridge regression
        ii. LASSO
5. Evaluate the best models on the test data (+ interpretation) *(point: 10)*
    a. MAE
    b. MAPE
    c. RMSE

# Thank you