# Introduction to Data Science Topic-3

- Instructor: Professor Henry Horng-Shing Lu,

Institute of Statistics, National Yang Ming Chiao Tung University, Taiwan

Email: henryhslu@nycu.edu.tw

- WWW: http://misg.stat.nctu.edu.tw/hslu/course/DataScience.htm
- Classroom: ED B27 (新竹市大學路1001號工程四館B27教室)
- References:

M. A. Pathak, Beginning Data Science with R, 2014, Springer-Verlag.

K.-T. Tsai, Machine Learning for Knowledge Discovery with R: Methodologies for Modeling, Inference, and Prediction, 2021, Chapman and Hall/CRC.

- Evaluation: Homework: 70%, Term Project: 30%
- Office hours: By appointment

# Course Outline

**10 Topics and 10 Homeworks:**

- **Introduction of Data Science**
- **Introduction of R and Python**
- **Getting Data into R and Python**
- **Data Visualization**
- **Exploratory Data Analysis**
- **Regression (Supervised Learning)**
- **Classification (Supervised Learning)**
- **Text Mining**
- **Clustering (Unsupervised Learning)**
- **Neural Network and Deep Learning**

# Getting Data into R

References:

Ch. 3, M. A. Pathak, Beginning Data Science with R, 2014, Springer-Verlag.

https://www.kaggle.com/code/dongdongxzoez/r-topic-3?scriptVersionId=105348041

# Reading Data – Case Study: Health Survey

Reading Data – Case Study: Health Survey

```
data <- read.csv("survey.csv") #read csv file
## View the first few rows of the data frame
head(data, 10)
```

```
##        sex height weight handedness exercise smoke
## 1   Female     68    158      Right     Some Never
## 2     Male     70    256       Left     None Regul
## 3     Male     NA    204      Right     None Occas
## 4     Male     63    187      Right     None Never
## 5     Male     65    168      Right     Some Never
## 6   Female     68    172      Right     Some Never
## 7     Male     72    160      Right     Freq Never
## 8   Female     62    116      Right     Freq Never
## 9     Male     69    262      Right     Some Never
## 10    Male     66    189      Right     Some Never
```

# Reading Data – Check Type and Structure

```
#class() function is used to determine the class or data type of an object
class(data)
```

```
## [1] "data.frame"
```

```
# Check the structure of the data frame
str(data)
```

```
## 'data.frame':    237 obs. of  6 variables:
##  $ sex       : chr  "Female" "Male" "Male" "Male" ...
##  $ height    : int  68 70 NA 63 65 68 72 62 69 66 ...
##  $ weight    : int  158 256 204 187 168 172 160 116 262 189 ...
##  $ handedness: chr  "Right" "Left" "Right" "Right" ...
##  $ exercise  : chr  "Some" "None" "None" "None" ...
##  $ smoke     : chr  "Never" "Regul" "Occas" "Never" ...
```

# Cleaning Up Data - Identifying Data Types

- A factor can be converted into an ordinal variable using the ordered() function.

```
> data$smoke
  [1] Never Regul Occas Never Never Never ...
Levels: Heavy Never Occas Regul

> data$smoke =ordered(data$smoke, levels=c('Never', 'Occas', 'Regul', 'Heavy'));data$smoke
 [1] Never Regul Occas Never Never Never ...
Levels: Never < Occas < Regul < Heavy
```

KS

what is ordinal in R ?

In R, an ordinal variable is a categorical variable that has a specific order or ranking associated with its categories. Ordinal variables differ from nominal variables, which have categories without any inherent order.

For example, consider a survey that asks respondents to rate their satisfaction with a product on a scale of "Very Dissatisfied," "Dissatisfied," "Neutral," "Satisfied," and "Very Satisfied." In this case, the variable "satisfaction" is ordinal because there is a clear order to the categories from least satisfied to most satisfied. The ordinal nature of this variable allows us to say that "Satisfied" is a higher level of satisfaction than "Dissatisfied," but it doesn't provide information about the exact difference in satisfaction levels.

Ordinal variables are often encountered in social sciences, psychology, and surveys where Likert scales or other ranking systems are used to measure attitudes, opinions, or perceptions. Analyzing ordinal data requires specific statistical methods designed for such data types, as they have characteristics that are distinct from continuous numerical variables. Common statistical analyses for ordinal data include ordinal logistic regression and non-parametric tests like the Mann-Whitney U test and the Kruskal-Wallis test.

In R, you can represent and work with ordinal variables using specialized packages like "ordinal" or by converting categorical variables into ordinal factors. The "ordinal" package provides tools for modeling and analyzing ordinal data. You can create an ordinal variable using the `ordered()` function in R, specifying the levels and their order.

# Cleaning Up Data -Check for unique

- We use the unique() command to get a list of unique values.

```
##Cleaning up data - Data entry error
unique(data$sex) #get unique value
```

```
## [1] "Female" "Male"   NA       "F"      "M"
```

```
unique(data$height)
```

```
##  [1] 68 70 NA 63 65 72 62 69 66 61 75 71 73 67 79 74 64 76 60 77 59
```

```
##Cleaning up data - Data entry error
unique(data$sex) #get unique value
```

```
## [1] "Female" "Male"   NA       "F"      "M"
```

```{r}
data$sex[data$sex == 'F'] = 'Female' # replace 'F" to 'Female'
data$sex[data$sex == 'M'] = 'Male'   # replace 'M' to 'Male'
unique(data$sex)
```

```
[1] "Female" "Male"   NA
```

- Replace any occurrences of 'F' and ‘M' in the "sex" column with 'Female‘ and ‘Male’. It's a common data preprocessing step used to ensure consistency and clarity in the representation of categories within a dataset. This process can be particularly useful when dealing with categorical variables that might have multiple representations for the same category.

KS

unique() function in R?

In R, the `unique()` function is used to find and return the unique values from a vector, array, or data frame. It is a helpful function for identifying distinct values within a dataset. Here's how to use the `unique()` function:

**Syntax:**

R Copy code

```
unique(x, incomparables = FALSE, fromLast = FALSE)
```

- `x`: The vector, array, or data frame from which you want to extract unique values.
- `incomparables`: A logical value indicating how to treat non-comparable elements (e.g., `NA` values). If set to `FALSE` (the default), `NA` values are treated as distinct unique values. If set to `TRUE`, `NA` values are removed.
- `fromLast`: A logical value that specifies whether to return the unique values in reverse order (from the last occurrence to the first). Default is `FALSE`.

# Check for Missing Values (NA or NaN)

```
# Count rows with complete cases (no missing values)
cat("Number of complete rows:", sum(complete.cases(data)), "\n")
```

```
## Number of complete rows: 207
```

```
# Count rows with missing values
cat("Number of rows with missing values:", sum(!complete.cases(data)), "\n")
```

```
## Number of rows with missing values: 30
```

KS

how many ways the missing values presents in the datasets

Missing values can be represented in datasets in various ways, depending on how the dataset is structured and how missing data is recorded. Here are some common ways that missing values can be represented in datasets:

1. **NA or NaN**: In R, missing values are often represented as "NA" (Not Available) for numerical data or "NaN" (Not a Number) for calculations that result in undefined values.
2. **Blank Cells**: In spreadsheet software like Microsoft Excel or Google Sheets, missing values are often represented as empty or blank cells.
3. **Placeholder Values**: Some datasets use specific placeholder values to indicate missing data. For example, a dataset might use "-9999" or "9999" to represent missing values.
4. **Question Marks or Symbols**: Missing values can be denoted using question marks ("?") or other symbols like asterisks ("*") in text-based datasets.
5. **Special Codes**: Datasets may use special codes or labels to indicate missing values. For example, "999" might represent missing data in a dataset where all other values are integers.
6. **Categorical Labels**: In categorical data, missing values might be represented by a specific category label such as "Unknown," "Not Applicable," or "N/A."
7. **Dots or Dashes**: Some datasets use dots (".") or dashes ("-") to indicate missing values in text data.
8. **Data Attributes**: Missing values can also be specified using data attributes or metadata, where a separate file or information in the dataset's documentation indicates which values are missing.

are missing.

9. **Sparse Data Structures**: In cases where the dataset is stored as a sparse data structure, missing values may be represented by the absence of a data point in a specific location.
10. **No Indication**: In some cases, missing values are not explicitly indicated in the dataset, and their presence is inferred from the context or analysis.

It's important to be aware of how missing values are represented in a dataset, as the choice of representation can affect data cleaning, imputation, and analysis processes. Additionally, understanding the specific representation of missing data is crucial for handling and interpreting the dataset accurately.

# Check for total Missing Values (NA or NaN)

### Missing Values

```r
# Count rows with complete cases (no missing values)
cat("Number of complete rows:", sum(complete.cases(data)), "\n")
```

```
## Number of complete rows: 207
```

```r
# Count rows with missing values
cat("Number of rows with missing values:", sum(!complete.cases(data)), "\n")
```

```
## Number of rows with missing values: 30
```

### Check for the total number of missing values in each column

```r
# Count rows with missing values
cat("Number of rows with missing values:", sum(!complete.cases(data$sex)), "\n")
```

```
## Number of rows with missing values: 1
```

```r
# Count rows with missing values
cat("Number of rows with missing values:", sum(!complete.cases(data$height)), "\n")
```

KS

complete.cases() function in R?

In R, the `complete.cases()` function is used to identify rows in a data frame or matrix that have complete cases, i.e., rows with no missing (NA) values. It returns a logical vector of the same length as the number of rows in the data structure, with each element set to `TRUE` for rows with no missing values and `FALSE` for rows containing at least one missing value.

**Syntax:**

R Copy code

```R
complete.cases(data)
```

- `data`: The data frame or matrix for which you want to identify complete cases.

**Example:**

R Copy code

```R
# Create a data frame with missing values
data <- data.frame(
  Name = c("Alice", "Bob", NA, "David"),
  Age = c(25, 30, NA, 35)
)

# Identify complete cases using complete.cases
complete_rows <- complete.cases(data)

# Print the result
print(complete_rows)
```

Output:

graphql Copy code

```
[1]  TRUE  TRUE FALSE  TRUE
```

In this example:

- We have a data frame `data` with two columns: "Name" and "Age," and it contains some missing values represented by `NA`.
- We use `complete.cases(data)` to check for complete cases. The resulting logical vector, `complete_rows`, has `TRUE` for rows without missing values (complete cases) and `FALSE` for rows containing missing values.

The `complete.cases()` function is useful when you want to filter a data frame to include only rows with complete data or when you need to perform operations that require complete cases, such as statistical analyses or modeling.

**Check for the Position of the Missing values in each columns**

```
###Cleaning up data - Missing value
which(is.na(data$sex)) #check where is the missing value
```

```
## [1] 137
```

```
which(is.na(data$height)) #check where is the missing value
```

```
##  [1]   3  12  15  25  26  29  31  35  58  68  70  81  83  84  90  92  96 108 121
## [20] 133 157 173 179 203 213 217 225 226
```

KS

which(is.na()) function in R

In R, you can use the `which(is.na())` combination of functions to identify the indices (positions) of missing (NA) values within a vector, matrix, or data frame.

Here's how it works:

1. `is.na(x)` is used to check for missing values within the data structure `x`. It returns a logical vector where `TRUE` indicates missing values (NA) and `FALSE` indicates non-missing values.
2. `which()` is used to find the indices (positions) where a condition is `TRUE` within a logical vector. In this case, it's used to find the indices where `is.na()` is `TRUE`.

KS

when to replace a missing dataset and when not

Deciding whether to replace missing data or leave it as-is in a dataset depends on several factors, including the nature of the data, the analysis you plan to perform, and the potential impact of missing values on your research or application. Here are some considerations to help you decide when to replace missing data and when not to:

**When to Replace Missing Data:**

1. **Missing Completely at Random (MCAR)**: If the missing data is completely at random and the likelihood of data being missing is unrelated to any variables, you may consider replacing missing values. In this case, imputation methods like mean imputation or regression imputation can be applied.
2. **Statistical Analysis**: For some statistical analyses, missing values can cause issues or bias the results. Replacing missing values with appropriate imputed values can help ensure the validity of your analysis.
3. **Machine Learning**: Many machine learning algorithms do not handle missing values well. In such cases, you may need to replace missing data with imputed values to make your data suitable for modeling.
4. **Preserving Data Integrity**: In cases where you want to maintain the structure and integrity of your dataset, replacing missing data with imputed values can be preferable.
5. **Data Visualization**: If you plan to create visualizations from your data, dealing with missing data can be challenging. Imputing missing values can make it easier to generate meaningful visualizations.

**When Not to Replace Missing Data:**

1. **Missing Not at Random (MNAR):** If you suspect that the reason for missing data is related to the value itself (e.g., people with high income are less likely to disclose their income), imputation may introduce bias. In such cases, it might be better to leave the data as-is and conduct sensitivity analyses.
2. **Small Sample Size**: If your dataset is already small and imputing missing data would further reduce the sample size, it may be better to perform analyses with the available data and acknowledge the limitations due to missing values.
3. **Transparency and Reporting**: In some cases, it's essential to report the proportion and nature of missing data as part of your analysis. Replacing missing data without acknowledging it can be misleading.
4. **Domain Knowledge**: Consider whether you have domain knowledge that suggests that missing data should not be replaced. In some fields, missing data might be informative and should not be imputed.
5. **Data Collection Costs:** If the cost or effort required to collect missing data is prohibitively high, it might be more practical to analyze the available data as-is.

Ultimately, the decision to replace missing data should be guided by the goals of your analysis, the characteristics of your dataset, and your understanding of the domain. It's important to carefully consider the potential impact of imputation on your results and to document your data preprocessing steps transparently.

# How to deal with the Missing Values in the Data?

KS

How to deal with the Missing Values in the Data?

Dealing with missing values in your data is an essential step in data preprocessing to ensure that your analyses and models are accurate and reliable. Here are some common strategies for handling missing values:

1. **Identify Missing Values:**
   - Use functions like `is.na()` in R or `pd.isnull()` in Python to identify missing values in your dataset.
2. **Understand the Nature of Missing Data:**
   - Determine whether missing data is Missing Completely at Random (MCAR), Missing at Random (MAR), or Missing Not at Random (MNAR). Understanding this can guide your handling strategy.

# Different Types of Data Imputation

3. **Data Imputation:**
   - **Mean/Median Imputation:** Replace missing numerical values with the mean or median of the column.
   - **Mode Imputation:** Replace missing categorical values with the mode (most frequent category) of the column.
   - **Regression Imputation:** Predict missing values using regression models based on other variables.
   - **K-Nearest Neighbors (K-NN) Imputation:** Replace missing values with values from the K-nearest neighbors.
   - **Multiple Imputation:** Generate multiple imputed datasets and analyze them separately, then combine results.
   - **Imputation using Machine Learning:** Train a machine learning model to predict missing values.

4. **Remove Rows/Columns:**
   - If missing data is extensive or cannot be reasonably imputed, consider removing entire rows or columns with missing values.
   - Be cautious with this approach, as it can lead to loss of valuable information.
5. **Imputation with Domain Knowledge:**
   - Use domain-specific knowledge to impute missing values. For example, if you have historical data, you may know that missing values should be zero or another specific value.
6. **Missing Data Indicators:**
   - Create a binary indicator variable that denotes whether a value is missing or not. This allows you to include missingness as a feature in your analysis.
7. **Data Sampling:**
   - In cases of MCAR or MAR, you can sample data with complete cases for certain analyses to avoid imputation.
8. **Advanced Techniques:**
   - Explore advanced imputation techniques like Random Forest imputation, Bayesian imputation, or deep learning-based imputation methods.
9. **Consider the Impact on Analysis:**
   - Be aware of how your chosen imputation method may impact the results of your analysis or modeling. Some methods can introduce bias.
10. **Documentation and Reporting:**
    - Clearly document how missing data was handled in your analysis. Be transparent in reporting the proportion of missing data and the methods used for imputation or handling.
11. **Validation:**
    - If you impute missing data, validate the imputed values to ensure they make sense in the context of your data.
12. **Consultation:**
    - Consider consulting with domain experts or statisticians, especially when dealing with complex missing data scenarios.

Remember that there is no one-size-fits-all approach to handling missing data, and the choice of strategy should depend on the nature of the data, the research question, and the impact on the analysis or modeling task.

# Replace continuous data with mean

```r
df$sex[df$sex == 'F'] = 'Female' # replace 'F" to 'Female'
df$sex[df$sex == 'M'] = 'Male'   # replace 'M' to 'Male'

#caculate mean of female height
female.height <- mean(df$height[which(df$sex == 'Female')], na.rm=T)
#caculate mean of male height
male.height <- mean(df$height[which(df$sex == 'Male')], na.rm=T)
#replace NA with mean of female & male height
df$height[which(df$sex == 'Female' & is.na(df$height))] = female.height
df$height[which(df$sex == 'Male' & is.na(df$height))] = male.height
```

# How to deal with the missing value in the categorical dataset?

Handling missing values in categorical data requires different strategies compared to numerical data. Here are several methods for dealing with missing values in categorical data:

1. **Imputation with Mode:**
   - Replace missing categorical values with the mode (most frequent category) of the respective column. This is a simple and often effective approach.
2. **Create a New Category:**
   - Introduce a new category label, such as "Missing" or "Unknown," to represent missing values. This allows you to retain information about the absence of data.
3. **Use Domain Knowledge:**
   - If you have domain expertise or knowledge about the data, you can manually assign missing values based on context. This is particularly useful when missing values can be reasonably inferred.
4. **Predictive Imputation:**
   - Train a machine learning model (e.g., decision tree or random forest) to predict missing categorical values based on other features. The model should be trained on rows with complete data.
5. **Multiple Imputation:**
   - Perform multiple imputation, where you create multiple imputed datasets with different imputed values for missing categories. This approach captures uncertainty associated with imputation.
6. **Hot Deck Imputation:**
   - Match each record with missing data to a similar record with complete data based on other attributes. Replace the missing value with the corresponding category from the matched record.
7. **Weighted Imputation:**
   - Assign weights to categories based on their frequency in the dataset. Use these weights to randomly sample and impute missing values. This approach preserves the distribution of existing data.
8. **K-Nearest Neighbors (K-NN) Imputation:**
   - Use the K-NN algorithm to find similar data points with complete information and impute missing values based on the majority category in their neighborhood.
9. **Regression Imputation:**
   - Fit a regression model where the categorical variable with missing data is the dependent variable, and other variables serve as predictors. Predict missing categories based on the model's coefficients.
10. **Consider a Separate Category:**
    - In some cases, it may be appropriate to consider missing data as a separate category if it carries meaningful information.
11. **Missing Data Indicator:**
    - Create a binary indicator variable that flags whether a value in the categorical column is missing or not. This can be useful if missingness carries information.
12. **Remove Rows:**
    - If missing data is extensive and cannot be reasonably imputed, consider removing rows with missing categorical values. Be cautious with this approach, as it may lead to a significant loss of data.
13. **Consult with Domain Experts:**
    - Collaborate with experts in the domain relevant to your data to make informed decisions about handling missing values.

The choice of method depends on the nature of your data, the reason for missingness, and the specific requirements of your analysis or modeling task. It's essential to document how you handled missing categorical values in your data preprocessing, as this information is critical for reproducibility and transparency.

# Replace categorical data with bootstrap

```r
bootstrap <- function(series){
    tb <- table(series[complete.cases(series)])
    prob <- tb / sum(tb)
    smpl <- sample(names(tb), prob=prob, size=sum(!complete.cases(series)))
    series[!complete.cases(series)] <- smpl
    return (series)
}
```

```r
df$smoke <- ordered(df$smoke, levels=c('Never', 'Occas', 'Regul', 'Heavy'))
df$smoke[!complete.cases(df)] <- names(which.max(table(df$smoke)))

df$exercise <- ordered(df$smoke, levels=c('None', 'Some', 'Freq'))
df$exercise[!complete.cases(df)] <- names(which.max(table(df$exercise)))

df$handedness <- bootstrap(df$handedness)
df$sex <- bootstrap(df$sex)
```

# Summary Data

In [7]:

```
cat("Numberof non NA row:", dim(df[complete.cases(df),])[1])
cat("\nNumberof non NA row:", dim(df[!complete.cases(df),])[1])
```

```
Numberof non NA row: 237
Numberof non NA row: 0
```

In [8]:

```
summary(df)
```

```
     sex                 height          weight        handedness
 Length:237         Min.   :59.00   Min.   :  0.0   Length:237
 Class :character   1st Qu.:65.27   1st Qu.:150.0   Class :character
 Mode  :character   Median :67.00   Median :169.0   Mode  :character
```

# Homework 3 (submitted to e3.nycu.edu.tw before Oct 11, 2023)

- Use R, Python, and suitable computer packages to analyze the data set with missing data (NA) that you select.
- Try to solve the missing value or another data error issue.
- Use Different Types of Data Imputation Methods for handling missing data in your datasets.
- Explain the results you obtain
- Discuss possible problems you plan to investigate for future studies

**Possible sources of open datasets:**

- UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.php)
- Kaggle Datasets (https://www.kaggle.com/datasets)

# Getting Data into Python

References:

Ch. 3, M. A. Pathak, Beginning Data Science with R, 2014, Springer-Verlag.

# Import dependencies

```
import pandas as pd
import numpy as np
from colorama import Fore, Back, Style
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split, KFold
from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import FunctionTransformer
```

# Reading Data – House Prices Prediction

```
df = pd.read_csv('../input/house-prices-advanced-regression-techniques/train.csv')
model = LinearRegression()
df.head(6)
```

| | Id | MSSubClass | MSZoning | LotFrontage | LotArea | Street | Alley | LotShape | LandContour | Utilities | ... | PoolArea | PoolQC | Fence | Mi |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 60 | RL | 65.0 | 8450 | Pave | NaN | Reg | Lvl | AllPub | ... | 0 | NaN | NaN | |
| **1** | 2 | 20 | RL | 80.0 | 9600 | Pave | NaN | Reg | Lvl | AllPub | ... | 0 | NaN | NaN | |
| **2** | 3 | 60 | RL | 68.0 | 11250 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | |
| **3** | 4 | 70 | RL | 60.0 | 9550 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | |
| **4** | 5 | 60 | RL | 84.0 | 14260 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | |
| **5** | 6 | 50 | RL | 85.0 | 14115 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | MnPrv | |

6 rows × 81 columns

# Fill continuous data with custom function

```
def fillMedian(data):

    data = data.select_dtypes(exclude=['object'])
    columns = list(data.columns)

    for col in columns:
        data[col].fillna(value=data[col].median(), inplace=True)

    return data
```

```python
def categorical_transformer(data):

    data = data.select_dtypes(exclude=['int64','float64'])

    columns = data.columns
    one_hot_cols = []
    label_cols = []
    for col in columns:
        data[col] = data[col].fillna(data[col].mode()[0])
        if len(data[col].unique()) < 10:
            one_hot_cols.append(col)
        else:
            label_cols.append(col)

    label_transformer = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
    one_hot_transformer = OneHotEncoder(handle_unknown="ignore")

    preprocessor = ColumnTransformer(
        transformers=[
            ("num", label_transformer, label_cols),
            ("cat", one_hot_transformer, one_hot_cols),
        ]
    )

    _data = preprocessor.fit_transform(data)
    return _data
```

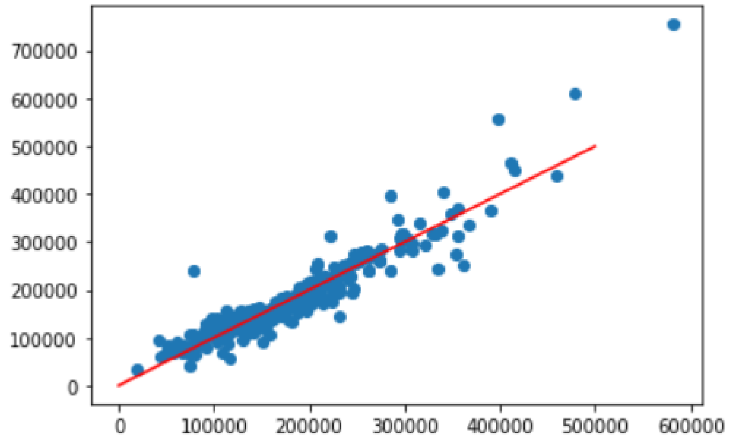
# Fit and Transform

```
X = df.drop(['SalePrice'], axis=1)
y = df.SalePrice

numeric_transformer = FunctionTransformer(fillMedian)
categorical_transformer = FunctionTransformer(categorical_transformer)
columns = X.columns

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, columns),
        ("cat", categorical_transformer, columns),
    ]
)

_X = preprocessor.fit_transform(X)
```

# Data Split

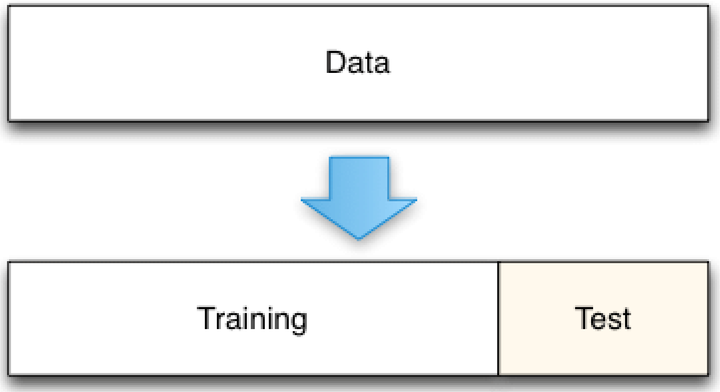
```
X_train, X_valid, y_train, y_valid = train_test_split(_X, y, test_size = 0.2, random_state=42)

model.fit(X_train, y_train)
pred = model.predict(X_valid)

grid = np.linspace(0, 500000, 1460)

plt.scatter(pred, y_valid)
plt.plot(grid, grid, "b-", c='red')
```

```
[<matplotlib.lines.Line2D at 0x7feaaa013c50>]
```




```
print(_X.shape)
np.isnan(_X).sum()
```

```
(1460, 236)
0
```

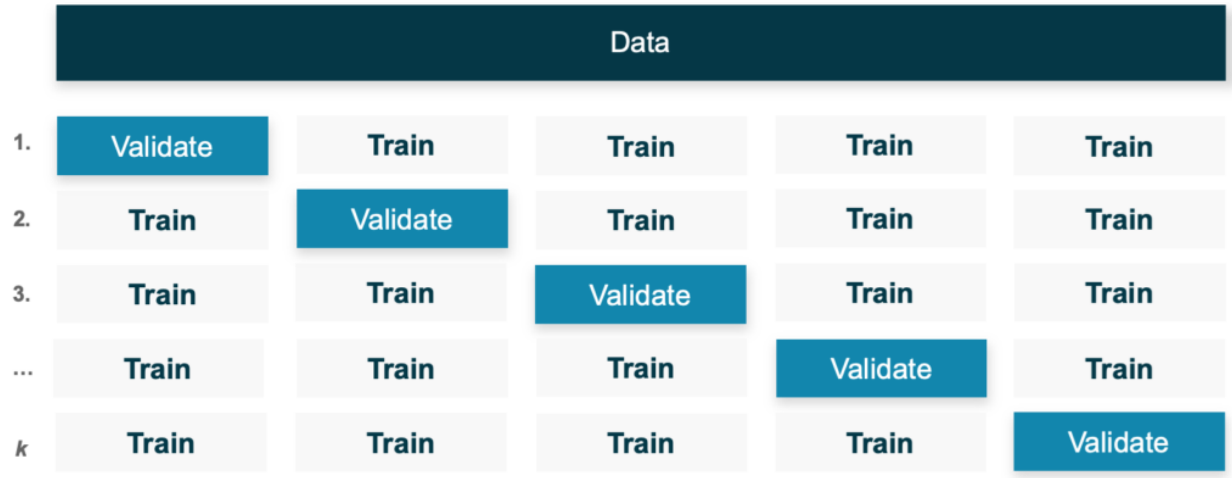


# Scoring MSE

```python
def scoring_mse(X_train, X_valid, y_train, y_valid, model):

    model.fit(X_train, y_train)
    preds = model.predict(X_valid)
    return mean_squared_error(y_valid, preds, squared=False)

scoring_mse(X_train, X_valid, y_train, y_valid, LinearRegression())
```

```
30754.592815992128
```

```
%%time

kf = KFold(n_splits=5, shuffle=True, random_state=1)
params = {}
score_list = []
for fold, (idx_train, idx_valid) in enumerate(kf.split(_X)):

    X_train = _X[idx_train]
    y_train = y[idx_train]

    model = LinearRegression(**params)

    model.fit(X_train, y_train)
    del X_train, y_train

    X_valid = _X[idx_valid]
    y_valid = y[idx_valid]
    y_valid_pred = model.predict(X_valid)
    rmse = mean_squared_error(y_valid, y_valid_pred, squared=False)
    del X_valid, y_valid

    print(f"Fold {fold}: rmse = {rmse:.5f}")
    score_list.append(rmse)

result_df = pd.DataFrame(score_list, columns=['rmse'])
print(f"{Fore.GREEN}{Style.BRIGHT}Average  mse = {result_df.rmse.mean():.5f}")
```




https://medium.com/@kn12/k-fold-cross-validation-using-python-code-e8e01039dc6

```
Fold 0: rmse = 32107.33766
Fold 1: rmse = 57351.50794
Fold 2: rmse = 26066.51942
Fold 3: rmse = 44514.71104
Fold 4: rmse = 22062.43021
Average  mse = 36420.50125
CPU times: user 575 ms, sys: 386 ms, total: 961 ms
Wall time: 248 ms
```