1) Handling Missing Values →

df.column.fillna()

fill with	when
1) mean/median	Data is numeric. mean → Normally distributed data. median → Skewed data.
2) mode	Categorical variable.

df.groupby('occupation')['Income'].transform(lambda x: x.mean())

2) Handle outliers →

Quantiles → The parts of data that each contain 25% of the data, low to high when sorted in ascending order.

data → Q_1, Q_2, Q_3, Q_4.

IQR → Inter quartile range → $Q_3 - Q_1$.

If (point < $q_1 - 1.5 \times iqr$) outlier.

if (point > or = $q_3 + 1.5\ iqr$) outlier.

a) Cap data → Replace outliers with q_1 & q_3 respectively.

b) log(data) → Compresses the data & brings the outliers close.

2) Scaling The Data →

a) Standardization → Converting the values to z-scores.

- Good for uniformly distributed data.
- Less sensitive to outliers.
- use in models that are sensitive to scale.

$$x_{std} = (x - \bar{x})/\sigma$$

→ distance away from the mean.

→ 1 unit away from the mean

→ # of units away from mean.

b) Min-Max Normalization → Convert the values b/w 0 & 1.

- Very sensitive to outliers.
- used in NN, Image transformation, feature engineering.

↳ Convert find values from [0-255] → [0, 1]

$$x_{min\text{-}max} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

SScaler = StandardScaler()
MScaler = MinMaxScaler()

x_std = SScaler.fit_transform(x)
x_min-max = MScaler.fit_transform(x)
```

3) Data Encoding → Convert categorical values into numeric values.

Label Encoding | Ordinal Encoding

a) Label Encoding → use when the categories have no inherent order.

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data = {e, w, ex, s, n}
encoder.fit_transform(data)
```

n-4 | e-2 | s-3 | w-4 → data = {2, 4, 4, 2, 1}

↳ data after encoding.

3) Ordinal Encoding → when the categories have inherent order b/w them.

ex →
data = { hs, b, m, phd, hs }
data_ordered = { hs, bs, ms, phd }

```
from sklearn.preprocessing import OrdinalEncoder
en = OrdinalEncoder( categories = data_ordered )
d_encoded = en.fit_transform(data)
```

d_encoded → { 1, 2, 3, 4, 1 }
(hs, bs, m, Phd, hs)

4) One hot Encoding → Convert categorical variables binary columns where each represents the occurance of a variable.

one-hot-encoder / Pd.get_dummies() } → Both do the same thing but with inherent differences.

a) OneHotEncoder → - use with sklearn & ML pipelines.

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder( sparse = False )
n = encoder.fit_transform( data.categorical_column )
```

(→ returns a numpy matrix.

b) pd.get_dummies() → use when performing explanatory data analysis.
- returns a pandas data frame.

5) Model Evaluation Strategy → Choosing the parts of dataset that will train & test the model for comparision with other models.

a) Train-Test split → Split the data set into two parts & use each part to train & test the dataset separately.

- high variance → high randomness.
- Split data may not be balanced.
- good for quick & easy selections.

6) Cross-Validation :- Data is split into multiple folds, for a certain # of repetations a different fold is selected as validation set & rest others as train set.

- Repetation reduces randomness.
- Dataset might not be balanced.
- General use for model selection.

7) K-fold cross validation :- - Data is divided into K sets.
- Repeated K times, for each K
  - a validation set is selected & model trained on rest of the data.
- Folds can be unbalanced.

8) Repeated K-fold :- Repeat K-fold validation multiple times.

- Reduced variance.
- Imbalanced splits.
- Increases model robustness.

9) Stratified-K-folds :- Split the data into K folds & makes sure that each split is balanced.

- Balanced dataset.
- good for classification tasks.

Each class in each split has equal # of elements.

10) Linear Regression :- Model the relationship b/w a set of variables by fitting a straight line into them.

- one of the simplest & widely used ml algorithms.

Procedure :-

$Y = mx + c$ ← need to find these two.

x	y	x^2	y^2	xy
Σx	Σy	Σx^2	Σy^2	Σxy

$$m = \frac{n\Sigma xy - (\Sigma x)(\Sigma y)}{n\Sigma x^2 - (\Sigma x)^2}$$

↳ with an unit change in x, how much does y change.

$$C = \frac{\Sigma y - m \Sigma x}{n}$$

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
model = LinearRegression()
model.fit(x, y)
          ↳ dataframe.        → Series.
ȳ = model.predict(x_test)
                  ↳ dataframe
Print( mean_squared_error(y, ȳ), r2_score(y, ȳ))
```

7) Linear Regression Metrics →

Residue → $y_i - \bar{y}_i$

Median

a) Absolute error →

MedianAE → median($|y_i - \bar{y}_i|$)

- Less sensitive to outliers.
- Robust to variance in errors.

Skewness

Mean

a) Mean squared error.
b) Mean absolute error.
c) Root mean squared error.

---

a) Mean Absolute Error → $\frac{1}{n}\sum_{i=1}^{n} |y_i - \bar{y}_i|$

- Less sensitive to outliers.
- All errors should be equally weighted.

b) **Mean squared error →** $\frac{1}{n} \times \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

- Sensitive to outliers.
- Good for model optimization.
- Penalizes larger errors more.

c) **Root Mean Squared Error →** $\sqrt{\frac{1}{n} \times \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$

- Similar properties to mean squared error.
- Brings error to y's units.

d) **R^2 →**

$$1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

- To determine how much variance $(y_i - \hat{y}_i)$ does the model explains, when compared with the average model.

↳ The lower the better.

↳ The higher the better.

8) Assumptions For Linear Regression →

1) **Linearity →** There should a linear relationship between the dependent & independent variable.

if not followed →

- Incorrect predictions.

2) **Independence →** The data points should be independent of each other.

- Misleading tests.

3) **Homoscedasticity →** The variance in errors should be similar across all levels of the input variable.

ex → In sales data, no higher errors for high priced item compared to normal priced item.

- Unreliable predictions at certain ranges.

4) Normality of Residue → The residue should be normally distributed.

$\downarrow (y_i - \hat{y}_i)$

– Important confidence intervals.

5) No Multicollinearity → The variables in the independent variables set should be independent of each-other.

– Less information gained from dependent variables.

Logistic Regression → Simple & computationally easy classification algorithm.

→ Classification → Task of putting/classifying an item into a certain category.

Linear regression → Continuous outcome.

Logistic regression → Discrete outcome.

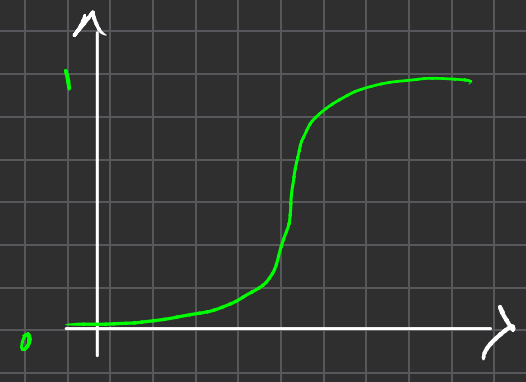
– Simple & efficient.
– Widely used.
– Probabilistic.

why? → True classification tasks.

what? → let $x = \sum w_i x_i + w_{bias}$

$$y = \frac{1}{1 + e^{-x}}$$

range $(y) = [0, 1]$ → good for interpreting as probability.



1) Confusion Matrix → A matrix used for evaluating model accuracy.

- Displays model predictions for categories.
- Shows how many predictions are correct & how many are wrong.

Actual \ Predicted	True	False
True	True +ve	False -ve
False	False +ve	True -ve

TP → Correctly classify as +ve item.

FP → Incorrectly classify as -ve item.

FN → Incorrectly classify as +ve item.

TN → Correctly classify as -ve item.

→ Metrics →

1) Accuracy →

$$\frac{\text{\# of correct predictions}}{\text{Total \# of predictions}} = \frac{TP + TN}{TP + FP + FN + TN}$$

2) Precision →

$$\frac{\text{\# of correct True predictions}}{\text{Total \# of True predictions}} = \frac{TP}{TP + FP}$$

3) Recall →

$$\frac{\text{\# of correct True predictions}}{\text{Total \# of true items}} = \frac{TP}{TP + FN}$$

4) F-1 Score → Balances Precision & Recall.

$$2 \times \text{Precision} \times \text{recall} / (\text{Precision} + \text{recall})$$

Regression Advanced

1) Adjusted R^2

a) **Problem with R^2** → Adding an useless column to data may increase R^2, thus giving inaccurate results.

going from $Y = w_1 x_1 \dots w_n x_n$

to

$Y = w_1 x_1 + \dots w_n x_n + w_{n+1} x_{n+1}$

might change R^2 value.

b) **Adjusted R^2** → Idea → Account for change in dimentionality.

↙ # of features.

$$\text{adjusted } R^2 = 1 - \frac{(1-R^2)(n-1)}{(n-m-1)}$$

range → $(-\infty, 1]$

n → # of rows.
m → # of columns.

high adjusted R^2 ⟶ Model explains more variance in dependent variable.

2) OLS Method of (Statsmodel → library)

↳ ordinary Least Square → estimating model parameters by minimizing sum of square errors.

- Richer model statistics than sklearn.

↳ offers additional features.

```
import statsmodel.api as sm
X_sm = sm.add_constant(X_train)   → adds a column for the bias co-efficients
model = sm.OLS(y_train, X_sm)
results = model.fit()
results.summary(), results.predict(X_sm)
```

Overfitting → Model performs well on training data but fails to generalize well on test data.

high variance & low bias.

Underfit → The data is too complex for the model & fails to perform good even on train data.

low variance & high bias.

→ **ways to reduce overfitting →**

1) use simpler model.
2) Regularization.
3) Early stopping → Stop once the validation error starts to increase.
4) Ensemble Methods.
5) Cross-validation.
6) Drop useless features.
7) Data Augmentation.

Regularization → A form/variant of regression with a penalty term to reduce overfitting.

1) **Ridge (L-2) →** loss = sum of sq of errors + sum of sq of weights
↓
Ridge-regression term.
- useful for limiting/minimizing the weights.
- Prevents the weights from increasing drastically.

2) **Lasso (L-1) →** loss = sum of sq errors + sum of absolute weights.
- useful to drop useless features.
→ lasso term

3) Elastic Net → loss = sum of sq errors + $\lambda(\alpha(L_2) + (1-\alpha)(L_1))$

↙ Controls strictness

↘ Balances regularization.

- Shrinks the weights smoothly.
- Removes less helpful variables.
- More stable than Ridge & Lasso alone.