Basic Data Preprocessing

working with data frame in pandas

```
# read from csv
df = pd.read_csv(“./customers.csv”, sep=“;”)
# inspect first 3 rows of dataframe
df.head(3)
# inspect the shape of the data
df.shape
# inspect column names
df.columns
# inspect the column data types
df.dtypes
# get summary of variable
df[“MonthlyCharges”].describe()
# get summary statistics of all numeric variables
df.describe()
# get the mean of the MonthlyCharges variable
df[“MonthlyCharges”].mean()
# get the unique categories of the InternetService variable
df[“InternetService”].unique()
# check if the InternetService variable only contains unique values
df[“InternetService”].is_unique()
# convert to numeric
pd.to_numeric(df_1[“Postal_Code”])
# convert to date format
pd.to_datetime(df_2[“dates”], format=“%d/%m/%Y”)
# sort based on Scores from big to small
df_1.sort_values(by=“Scores”, ascending=False)
# select row 2 and column 3
df_sub = df.iloc[1, 2]
# select rows 2, 5, 6 and column 4, 5
df_sub = df.iloc[[1, 4, 5], [3, 4]]
# select rows 10 until 30 and column 3 until 5
df_sub = df.iloc[10:30, 3:5]
# select the gender and tenure columns
df_sub = df[[“gender”, “tenure”]]
# select all the rows where gender equals Male
df_sub = df[df[“gender”] == “Male”]
# select all the rows where tenure equals 1, 2 or 3
df_sub = df[df[“tenure”].isin([1, 2, 3])]
```

```
# select all the rows where gender equals Male and OnlineSecurity equals Yes
df_sub = df[(df[“gender”] == “Male”) & (df[“OnlineSecurity”] == “Yes”)]
# group by gender and calculate mean of monthlycharegs for them
df.groupby(“gender”)[“MonthlyCharges”].mean()
# group by gender and apply function
agg_dict = {“tenure”:[np.mean, np.sum], “MonthlyCharges”:[np.min, np.max]}
df.groupby(“gender”).agg(agg_dict)
```

	MonthlyCharges		tenure	
	amin	amax	mean	sum
gender				
Female	18.40	118.75	32.244553	112469
Male	18.25	118.35	32.495359	115521

```
# merge two dataframe
pd.merge(left=df_1, right=df_2, on=key, how=“outer”/“inner”/“left” / ”right”)
# Drop every duplicate row,The subset parameter allows you to drop duplicates based on a subset of columns instead of all columns
df.drop_duplicates(subset=[columns])
# Drop one or multiple columns from a DataFrame
df.drop(columns=[columns])
# Define a function that will be applied on the specified column of each row and call it in theapply() function:
def replace_negative_values(column_obs):
  if column_obs < 0:
    return(0)
  else:
    return(column_obs)
df_1[“Spending”].apply(replace_negative_values)
# Use the lambda expression in the apply() function
df_1[“Spending”].apply(lambda x: 0 if x < 0 else x)
```

EDA: STATISTICAL DATA EXPLORATION

Univariate summary statistics:

- Mean
- Median
- Mode
- Interquartile range: $\mathrm{IQR} = \mathrm{Q}_3 - \mathrm{Q}_1$
- Standard deviation $s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$

Bivariate summary statistics:

- Pearson's correlation coefficient: $r = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2(y_i-\bar{y})^2}}$
  - strength of linear relationship between X and Y
  - $r \in [-1, 1]$

 # calculate the median of tenure

```
np.median(df["tenure"])
```

 # calculate the first quartile such that 25% of the observations are lower than the first quartile

```
np.percentile(df["tenure"], q=25)
```

 # calculate the minimum of MonthlyCharges

```
np.min(df["MonthlyCharges"])
```

 # calculate the range of MonthlyCharges(difference between the maximum and minimum)

```
np.ptp(df["MonthlyCharges"])
```

 # calculate the covariance between this two

```
np.cov([df["MonthlyCharges"], df["tenure"]])
```

 # calculate the covariance matrix for all numeric variables

```
df.cov(numeric_only=True)
```

Covariance Formula

For Population

$$\mathrm{Cov}(x,y) = \frac{\Sigma\,(x_i - \bar{x}) * (y_i - \bar{y})}{N}$$

For Sample

$$\mathrm{Cov}(x,y) = \frac{\Sigma\,(x_i - \bar{x}) * (y_i - \bar{y})}{(N-1)}$$

 # calculate the correlation between this two (standardizes the covariance)

```
np.corrcoef([df["SeniorCitizen"], df["tenure"]])
```

 # calculate the correlation matrix for all numeric variables

```
df.corr(numeric_only=True)
```

: measures how much each category occurred in a categorical variabl

```
df["Partner"].value_counts()
```

```
No    3641
Yes   3402
Name: Partner, dtype: int64
```

 # measures the relative proportion of each category of a categorical variable

```
df["Partner"].value_counts(normalize=True)
```

 # calculate the correlation matrix for all numeric variables

```
df["Partner"].mode()
```

 # crosstab is a table that is used to aggregate and jointly display the distribution of two or more categorical variables

```
pd.crosstab(df["gender"], df["Partner"])
```

Data cleaning

1.missing value

```
# check if the InternetService variable have any null
df["InternetService"].isnull()
# see how many null this variable has
df["InternetService"].isnull().sum()
# Drop every row with one or more NaNs
df_1.dropna()
# Drop every row with NaNs for every column
df_1.dropna(how="all")
# Drop rows where the 'InternetService' column has NaN
df.dropna(subset=["InternetService"], inplace=True)
# Impute missing data with user-defined values
column_imputations = {"Name": "UNKNOWN","Spend": 0}
df_1.fillna(value=column_imputations)
```

it is good to add flag to indicate imputation for one row

Impute with constant from training set → avoid data leakage

Table: Missing values: imputation

Height	→	Height	Height_missing
64		64	FALSE
73		73	FALSE
59		59	FALSE
NA		65	TRUE
64		64	FALSE

2.outlier

Types of outliers:

- Valid vs invalid observations
  e.g. income of $1,000,000 vs age of user is 120 years
- Univariate vs multivariate outliers:
  outlying on 1 dimension vs outlying on multiple dimensions

2,1: Detecting univariate outliers:

- Numerically:
  - Min and max values
  - Z-scores: $z_i = \frac{x_i - \mu}{\sigma}$
    outlier if $|z_i| > 3$
- Graphically:
  - Histograms
  - Boxplots:
    outlier if observation outside whiskers of box

```
# Min and max values
min_value = df["InternetService"].min()
max_value = df["InternetService"].max()
# Z-scores calculation

mean_value = df["InternetService"].mean()
 std_dev = df["InternetService"].std()
 df['z_score'] = (df["InternetService"] - mean_value) / std_dev

# Plotting Histograms
df["InternetService"].hist(bins=20)
# Plotting Boxplots
df.boxplot(column=["InternetService"])
```

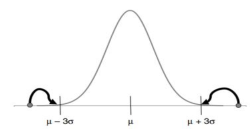
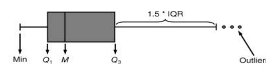


Figure: Z-score



2,1: Detecting multivariate outliers:

```
from sklearn.ensemble import IsolationForest
# Fit Isolation Forest model
clf = IsolationForest(contamination=0.1)  # contamination is the proportion of outliers
clf.fit(X)
# Predict outliers
outliers = clf.predict(X)
# -1 indicates outlier, 1 indicates normal
print(outliers)
```

Only apply this to the training set

Handling: Invalid observations: treat outlier as missing value Valid observations: truncation → impose lower and upper limit on values (+ indicator), based on ... Z-score: |zi |= 3 Expert opinion

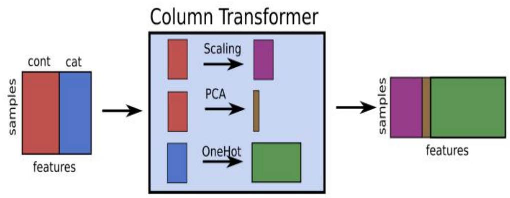
Nonparametric models (e.g. DT, NN, SVM) often insensitive to outliers
Parametric models (e.g. LinR, LogR) often sensitive to outliers.
(Nonparametric models are a type of statistical model that do not assume a specific form for the underlying data distribution.)

3.STRING MANIPULATION

```
# Apply the replacement using regex on the 'PhoneNumber' column
df['PhoneNumber'] = df['PhoneNumber'].str.replace(pattern, replacement, regex=True)
```

Basic Data Preprocessing

FEATURE ENGINEERING



Feature engineering techniques:

- Categorical predictors
  - dummy encoding
  - integer encoding
- Numeric predictors
  - feature scaling
  - feature transformations
  - feature interactions

CATEGORICAL DATA: ENCODING

Table: Integer encoding `drinks`

Original value	Integer variable
"not at all"	0
"rarely"	1
"socially"	2
"often"	3
"very often"	4
"desperately"	5

Table: Integer encoding `has_link`

Original value	Integer variable
FALSE	0
TRUE	1

```
# Integer Encoding
df['Category_Encoded'] = df['Category'].astype('category').cat.codes

# Define the order of the categories
category_order = {'High School': 1, 'Bachelor': 2, 'Master': 3, 'PhD': 4}

df['EducationLevel_Encoded'] = df['EducationLevel'].map(category_order)
```

Table: Dummy encoding `status`

Original value	Dummy variables			
	is_single	is_available	is_seeing_someone	is_married
"single"	1	0	0	0
"available"	0	1	0	0
"seeing someone"	0	0	1	0
"married"	0	0	0	1
"unknown"	0	0	0	0

```
# One-Hot Encoding using get_dummies
df_encoded = pd.get_dummies(df, columns=['Category'], prefix='Category')
```

Nominal features with high cardinality (> 10): select OHE categories with top frequency

Need same categories in train & test set

Be aware of the dummy variable trap!

NUMERICAL DATA: FEATURE SCALING

Not needed for linear/logistic regression, but scaling makes coefficients more interpretable

Necessary for models with penalization term (ridge/lasso)

Important (but not necessary) for models based on Euclidean distances (SVM, KNN)

Critical when performing PCA

Tree-based models scale-invariant

Scaling: making sure all numerical features have same scale

Two methods:

1. Normalization (min-max scaling)
   - values are shifted and rescaled such that they end up ranging from 0 to 1
   - $x_{new} = \frac{x_{old} - \min(x_{old})}{\max(x_{old}) - \min(x_{old})}$
2. Standardization (standard scaling):
   - values are centered and rescaled to fit a standard normal distribution $\mathcal{N}(\mu = 0, \sigma = 1)$
   - $x_{new} = \frac{x_{old} - \mu}{\sigma}$

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# 1. Normalization (Min-Max Scaling)
min_max_scaler = MinMaxScaler()
df_normalized = pd.DataFrame(min_max_scaler.fit_transform(df), columns=df.columns)
# 2. Standardization (Standard Scaling)
standard_scaler = StandardScaler()
df_standardized = pd.DataFrame(standard_scaler.fit_transform(df), columns=df.columns)
```