23CSE301 Machine Learning

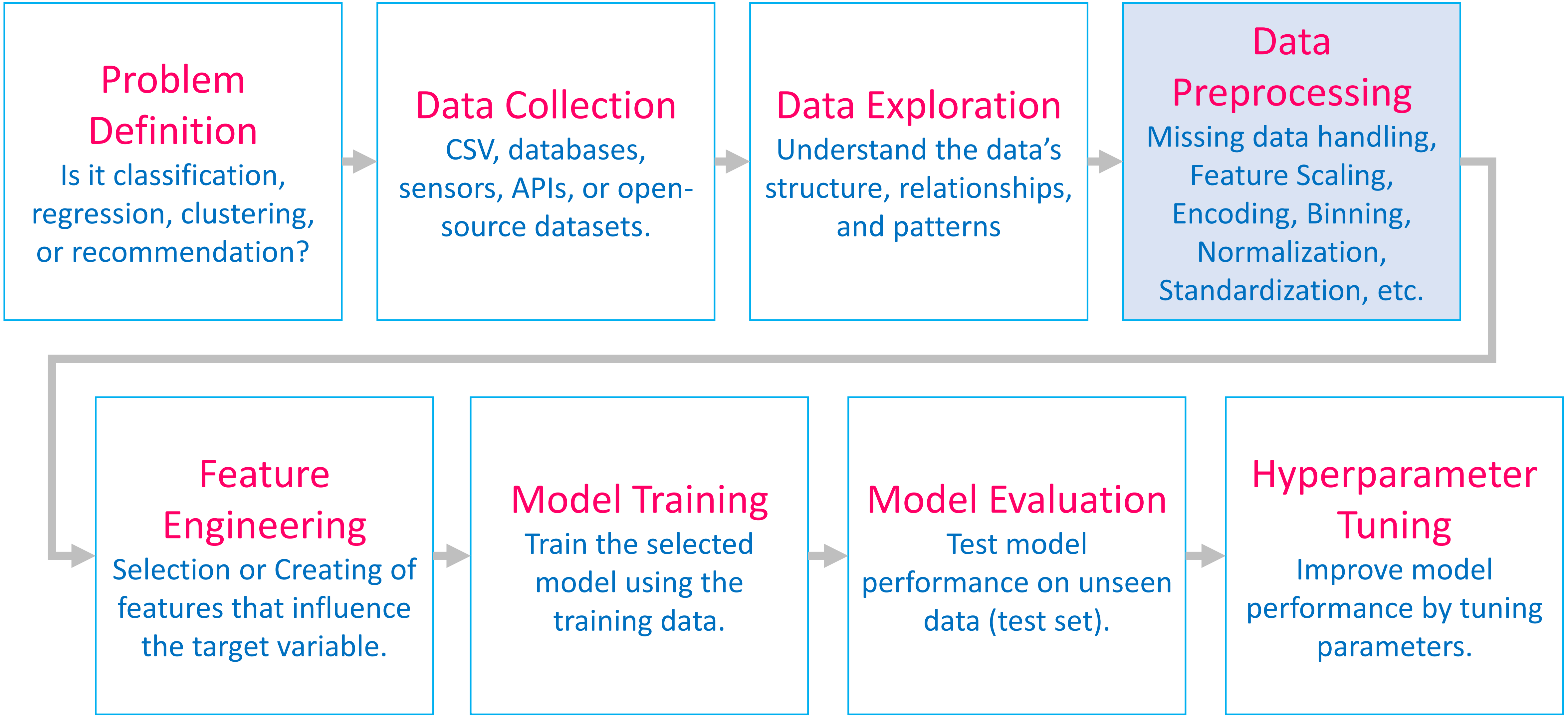
V Sem. CSE B

Practical – Week 3

Course Instructor: Dr. M. Anbazhagan

Pract. #	Experiment Title
P1-P3	• Introduction: Python, Pandas (scikit learn and other libraries) • Pre-processing: Dataset selection, Exploratory Data analysis and Feature engineering; Introduction to Colab/Jupyter Notebook, Pandas(Data Frames); Data Selection (iloc, loc); Sorting, Grouping merge, join, concat; Crosstab; Missing data treatment(fillna, dropna), Converting categorical values, Visualization(Line chart, Bar Chart, Pie chart, Scatter plot, Box plot); Distributions; Summary statistics.
Lab 1 Evaluation (P1 to P3)	
P4	Dimensionality Reduction Technique: PCA
P5	Feature Selection
P6	Regression Algorithms: Linear Regression
P7	Regression Algorithms: Logistic Regression
P8	Classification Algorithms: Decision Tree Classifier
	Classification Algorithms: K-Nearest Neighbor Classifier
Lab 2 Mid-Term exam (P1 to P8)	
P9	Classification Algorithms: Random Forest Classifier, ensemble learning.
P10	Classification Algorithms: Support Vector Machines
P11	Classification Algorithms: Perceptron
P12	Clustering: 1. K-Means Clustering 2. Agglomerative Clustering
Lab 3 Evaluation (P1 to P12)	

End-to-End Machine Learning Pipeline



Handling Missing Data

Handling missing data is a crucial step in data preprocessing. Here are six commonly used methods:

Simple Imputer

Replaces missing values with a constant or a statistical value like the mean, median, or mode of the column

Forward Fill (Pad)

Propagates the last valid observation forward to fill gaps

Backward Fill

Uses the next valid observation to fill missing values

KNN Imputer

Fills in missing values using the average of the nearest neighbors' values

Model-Based Imputation

Trains a predictive model to estimate missing values based on other features

Deletion

Removes rows (or columns) with missing values

Handling Missing Data

Simple Imputer

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')  # or 'median', 'most_frequent‘
data_imputed = imputer.fit_transform(data)
```

KNNImputer

```
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5)
data_imputed = imputer.fit_transform(data)
```

Forward Fill

```
data.fillna(method='ffill', inplace=True))
```

Backward Fill

```
data.fillna(method='bfill', inplace=True)
```

Deletion

```
data.dropna(inplace=True)
data.dropna(inplace=True, axis=1)
```

The number 5 refers to the parameter n_neighbors, which specifies how many nearest neighbors to use

fit(): Learns the parameters from the data
transform(): Applies those learned parameters to modify the data
fit_transform(): Does both.

The parameter axis=1 specifies that the operation should be applied to columns.

Feature Scaling

A preprocessing technique to normalize the range of independent variables in a dataset in order to ensure that all features contribute equally to the model's performance, especially for algorithms that are sensitive to the scale of data

Min-Max Scaling (Normalization)

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
```

Min-Max Scaling (Normalization)

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
```

Unit Vector Scaling (L2 normalization)

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer(norm='l2')
normalized_data = normalizer.fit_transform(data)
```

Feature Scaling

	rooms	age	distance	tax_rate	price
37	3.060495	19.465333	3.636863	743.210385	17.326524
82	8.216841	65.544165	0.077033	770.364288	45.607479
36	6.313295	32.757226	3.582223	776.714338	46.531825
39	6.295292	59.498401	1.042686	241.616781	26.833420
96	6.444180	97.328045	2.424691	464.918301	40.023571

	rooms	age	distance	tax_rate	price
77	0.518946	0.671474	0.081489	0.000000	0.318847
34	0.769739	0.086915	0.453326	0.949942	0.828764
45	0.424841	0.172656	0.148119	0.314533	0.370480
62	0.338417	0.913080	0.372966	0.368677	0.122741
73	0.935681	0.911180	0.217402	0.832548	0.231647

	rooms	age	distance	tax_rate	price
0	0.016181	0.101534	0.011399	0.992177	0.069881
1	0.008838	0.035074	0.003197	0.998904	0.029542
2	0.033224	0.061318	0.012298	0.994280	0.079957
3	0.030308	0.125935	0.007444	0.986454	0.100374
4	0.022780	0.380479	0.002628	0.917854	0.110697

	rooms	age	distance	tax_rate	price
40	0.932158	0.667452	0.622709	-1.311531	2.117170
44	-1.521305	0.554471	-0.361938	-1.410143	-2.210879
92	-0.662014	-1.369138	0.018096	-1.343431	-1.267112
64	1.014117	-0.513347	-0.968873	-0.511602	-0.074657
57	-0.227271	0.240651	-0.292687	-1.056043	-0.550654

Encoding

Feature encoding is the process of converting categorical data into a numerical format so that machine learning algorithms can process it

Label Encoding (for ordinal data)

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['color_encoded'] = le.fit_transform(df['color'])
```

One-Hot Encoding (for nominal data)

```
pd.get_dummies(df['color'], prefix='color')
```

Ordinal Encoding (when order matters)

```
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder(categories=[['low', 'medium', 'high']])
df['priority_encoded'] = encoder.fit_transform(df[['priority']])
```

Label Encoding

	city	education	experience_level	salary_range	performance
0	Mumbai	Graduate	Junior	Low	85
1	Delhi	Post-Graduate	Senior	High	92
2	Bangalore	Graduate	Mid	Medium	78
3	Chennai	High School	Junior	Low	65
4	Mumbai	Post-Graduate	Senior	High	95

	city	education	experience_level	salary_range	performance	education_encoded	experience_encoded	salary_encoded
0	Mumbai	Graduate	Junior	Low	85	0	0	1
1	Delhi	Post-Graduate	Senior	High	92	2	2	0
2	Bangalore	Graduate	Mid	Medium	78	0	1	2
3	Chennai	High School	Junior	Low	65	1	0	1
4	Mumbai	Post-Graduate	Senior	High	95	2	2	0
5	Delhi	Graduate	Mid	Medium	82	0	1	2
6	Pune	High School	Junior	Low	70	1	0	1
7	Bangalore	Graduate	Senior	High	88	0	2	0

Onehot Encoding

	city	education	experience_level	salary_range	performance
0	Mumbai	Graduate	Junior	Low	85
1	Delhi	Post-Graduate	Senior	High	92
2	Bangalore	Graduate	Mid	Medium	78
3	Chennai	High School	Junior	Low	65
4	Mumbai	Post-Graduate	Senior	High	95

	experience_level	salary_range	performance	city_Bangalore	city_Chennai	city_Delhi	city_Mumbai	city_Pune	edu_Graduate	edu_High School	edu_Post-Graduate
0	Junior	Low	85	False	False	False	True	False	True	False	False
1	Senior	High	92	False	False	True	False	False	False	False	True
2	Mid	Medium	78	True	False	False	False	False	True	False	False
3	Junior	Low	65	False	True	False	False	False	False	True	False
4	Senior	High	95	False	False	False	True	False	False	False	True

Label Encoding

	city	education	experience_level	salary_range	performance
0	Mumbai	Graduate	Junior	Low	85
1	Delhi	Post-Graduate	Senior	High	92
2	Bangalore	Graduate	Mid	Medium	78
3	Chennai	High School	Junior	Low	65
4	Mumbai	Post-Graduate	Senior	High	95

	city	education	experience_level	salary_range	performance
0	Mumbai	Graduate	Junior	0.0	85
1	Delhi	Post-Graduate	Senior	2.0	92
2	Bangalore	Graduate	Mid	1.0	78
3	Chennai	High School	Junior	0.0	65
4	Mumbai	Post-Graduate	Senior	2.0	95
5	Delhi	Graduate	Mid	1.0	82
6	Pune	High School	Junior	0.0	70
7	Bangalore	Graduate	Senior	2.0	88

Binnig

Binning is the process of converting continuous numerical variables into discrete categories or bins

Equal-width binning

```
age_data['age_bins_equal'] = pd.cut(age_data['age'], bins=4, labels=['Young', 'Adult', 'Middle-aged', 'Senior'])
```

	age	income	age_bins_equal	age_bins_quantile
0	22	30000	Young	Q1
1	25	45000	Young	Q1
2	30	60000	Young	Q1
3	35	75000	Adult	Q2
4	40	80000	Adult	Q2

Equal-frequency binning

```
age_data['age_bins_quantile'] = pd.qcut(age_data['age'], q=4, labels=['Q1', 'Q2', 'Q3', 'Q4'])
```

	age	income	age_bins_equal	age_bins_quantile
0	22	30000	Young	Q1
1	25	45000	Young	Q1
2	30	60000	Young	Q1
3	35	75000	Adult	Q2
4	40	80000	Adult	Q2

Custom Binning

Custom binning

```
custom_bins = [0, 30, 50, 70, 100]
custom_labels = ['Youth', 'Young Adult', 'Middle Age', 'Senior']
age_data['age_bins_custom'] = pd.cut(age_data['age'], bins=custom_bins, labels=custom_labels)
```

	age	income	age_bins_equal	age_bins_quantile	age_bins_custom
0	22	30000	Young	Q1	Youth
1	25	45000	Young	Q1	Youth
2	30	60000	Young	Q1	Youth
3	35	75000	Adult	Q2	Young Adult
4	40	80000	Adult	Q2	Young Adult

Exercise 1 - Week 3

- Data Preprocessing
  - You will practice all the preprocessing techniques learned in today's lab session (also listed below) using real datasets from the Seaborn library. This exercise is designed to simulate real-world data preprocessing scenarios.
    - Handling Missing Values
    - Feature Scaling
    - Encoding
    - Binning
  - You will practice all data preprocessing techniques (missing data handling, feature scaling, encoding, binning, and normalization) using three real-world Seaborn datasets: Tips, Flights, and Titanic. The exercise progresses from basic data exploration to building a complete preprocessing pipeline, requiring you to justify your method choices and analyze trade-offs