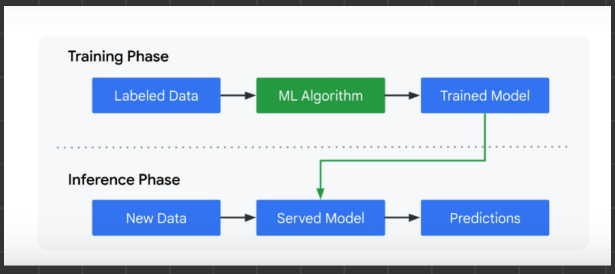
GCP → Improve Data Exploratory Data Analysis

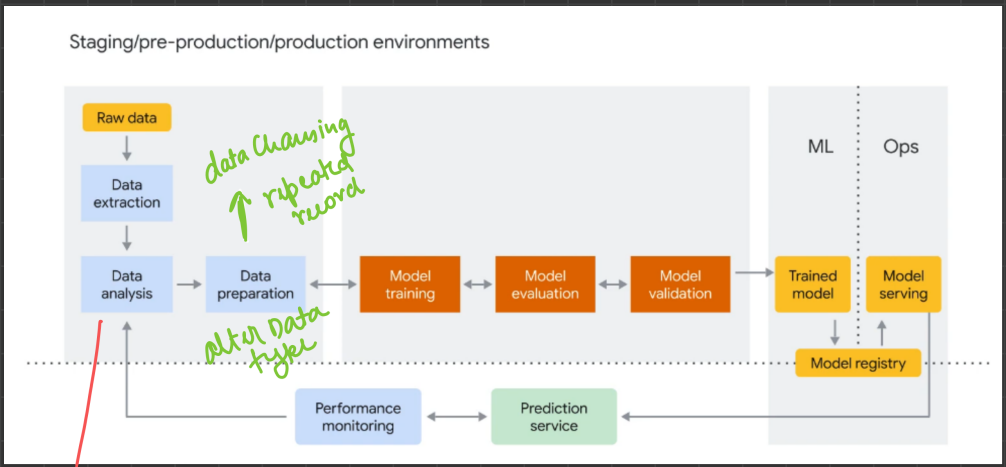
- Improve the quality of our Data
- Explore our Data by performing Data Analysis.

Here we learn → Perform
- → Exploratory Data Analysis
- → Categorize missing data strategies
- → Improve Data quality.

Improve Data quality

Two phase of Machine learning
- → Training Phase
- → Inference Phase





We can use EDA

Exploratory Data Analysis

which include

Basic Sample Statistics

graphic

To get information About Dataset

We look for different aspects

data distribution

Outlier

Anomalies

Trends

Model can handle type features → Categorical

→ Numerical

or mixed

Determining Data quality Levels

Accurate

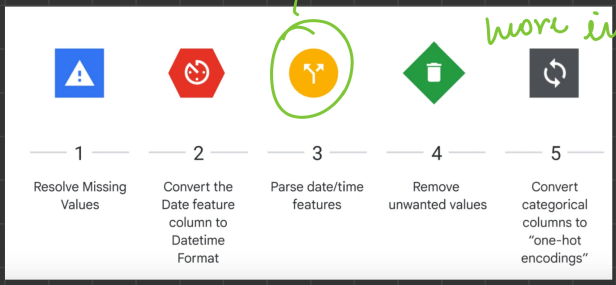
Complete ← Quality Data → Timely

Consistent

Timely
↳ Time when information is Expected
VS
When it is readily available to use.

For Example
Time = Data Capture — Real world Event Being Captured.

Parse Date/Time feature → That allows to create more insight in your Data




```
[5]: df_transport = pd.read_csv( `../data/transport/untidy_vehicle_data.csv` )
     df_transport.head() # Output the first five rows.
```

[5]:

	Date	Zip Code	Model Year	Fuel	Make	Light_Duty	Vehicles
0	10/1/2018	90000	2006	Gasoline	OTHER/UNK	NaN	1.0
1	10/1/2018	NaN	2014	Gasoline	NaN	Yes	1.0
2	NaN	90000	NaN	Gasoline	OTHER/UNK	Yes	Nan
3	10/1/2018	90000	2017	Gasoline	OTHER/UNK	Yes	1.0
4	10/1/2018	90000	<2006	Diesel and Diesel Hybrid	OTHER/UNK	No	55.0

```
In [19]: # TODO 2a
         df_transport.[`Date`] = pd.to_datetime(df_transport[`Date`],
                                                format= %m/%d/%Y)
In [20]: # TODO 2b
         df_transport.info() # Date is now converted
        <class `pandas.core.frame.DataFrame`>
         RangeIndex: 999 entries, 0 to 998
         Data columns (total 7 columns):
         Date            999 non-null datetime64[ns]
         Zip Code        999 non-null object
         Model Year      999 non-null object
         Fuel            999 non-null object
         Make            999 non-null object
         Light_Duty      999 non-null object
         Vehicles        999 non-null float64
         dtype: datetime64[ns](1), float64(1), object(5)
         memory usage: 54.8+ KB
```

Can Convert feature Column to Date Time

Help to Put time Series related application

```
In [21] df_transport[`year`] = df_transport[`Date`].dt.year
        df_transport[`month`] = df_transport[`Date`].dt.month
        df_transport[`day`] = df_transport[`Date`].dt.day
        #df[`hour`] = df[`date`].dt.hour - you could use if your
        #df[`minute`] = df[`date`].dt.minute - you could use this if
        df_transport.info()
        <class `pandas.core.frame.DataFrame`>
        RangeIndex: 999 entries, 0 to 998
        Data columns (total 10 columns):
        Date           999 non-null datetime64[ns]
        Zip Code       999 non-null object
        Model Year     999 non-null object
```

issue (3)

Unwanted characters - Model Year

Let's investigate a bit more of our data by using the .groupby() function.

```
In [9]: grouped_data = df_transport.groupby([`Zip Code`,`Model Year`,`Fuel`,`Make`,`Light_Duty`,`Vehicles`])
        df_transport.groupby(`Fuel`).first() # Get the first entry for each month.

Out [9]:
```

Fuel	Date	Zip Code	Model Year	Make	Light_Duty	Vehicles
Battery Electric	10/1/2018	90000	<2006	OTHER/UNK	No	4.0
Diesel and Diesel Hybrid	10/1/2018	90000	<2006	OTHER/UNK	No	55.0
Flex-Fuel	10/14/2018	90001	2007	Type_A	Yes	78.0
Gasoline	10/1/2018	90000	2006	OTHER/UNK	Yes	1.0
Hybrid Gasoline	10/24/2018	90001	2009	OTHER/UNK	Yes	18.0
Natural Gas	10/25/2018	90001	2009	OTHER/UNK	No	2.0
Other	10/8/2018	90000	<2006	OTHER/UNK	Yes	6.0
Plug-in Hybrid	11/2/2018	90001	2012	OTHER/UNK	Yes	1.0

"<" this Sign

① Make this year Bucket

② Remove "<" Sign which will Cause data loss.

Categorical data - "Yes/No"

```
df_transport = pd.read_csv(`../data/transport/untidy_vehicle_data.csv`)
df_transport.head() # Output the first five rows.
```

	Date	Zip Code	Model Year	Fuel	Make	Light_Duty	Vehicles
0	10/1/2018	90000	2006	Gasoline	OTHER/UNK	NaN	1.0
1	10/1/2018	NaN	2014	Gasoline	NaN	Yes	1.0
2	NaN	90000	NaN	Gasoline	OTHER/UNK	Yes	Nan
3	10/1/2018	90000	2017	Gasoline	OTHER/UNK	Yes	1.0
4	10/1/2018	90000	<2006	Diesel and Diesel Hybrid	OTHER/UNK	No	55.0

Categorical Value

Make One hot coding to Convert the categorical data To Digit

One-hot encoding

Ocean_Proximity	Ocean_Proximity=INLAND	Ocean_Proximity=NEAR BAY	Ocean_Proximity=OCEAN
INLAND	1	0	0
NEAR BAY	0	1	0
OCEAN	0	0	1

ML Algo → Can't work with Categorical Data directly

Instead generate 1 Boolean Column for each Category or class

Improve Data quality :->

↳ How to Convert date feature Columns to a date/time feature.

↳ Create One-hot Coding feature

Improving Data Quality

Learning Objectives

1. Resolve missing values
2. Convert the Date feature column to a datetime format
3. Rename a feature column, remove a value from a feature column
4. Create one-hot encoding features
5. Understand temporal feature conversions

Introduction

Recall that machine learning models can only consume numeric data, and that numeric data should be "1"s or "0"s. Data is said to be "messy" or "untidy" if it is missing attribute values, contains noise or outliers, has duplicates, wrong data, upper/lower case column names, and is essentially not ready for ingestion by a machine learning algorithm.

This notebook presents and solves some of the most common issues of "untidy" data. Note that different problems will require different methods, and they are beyond the scope of this notebook.

Each learning objective will correspond to a *#TODO* in this student lab notebook -- try to complete this notebook first and then review the solution notebook.

Start by importing the necessary libraries for this lab.

Import Libraries

```
# Importing necessary tensorflow library and printing the TF version.
import tensorflow as tf

print("TensorFlow version: ",tf.version.VERSION)
```

✓ 13.1s Python

```
WARNING:tensorflow:From c:\Users\ankit\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\losses.py:29

TensorFlow version:  2.15.0
```

```
import os
import pandas as pd  # First, we'll import Pandas, a data processing and CSV file I/O library
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

✓ 3.1s Python

Load the Dataset

The dataset is based on California's Vehicle Fuel Type Count by Zip Code report. The dataset has been modified to make the data "untidy" and is thus a synthetic representation that can be used for learning purposes.

Let's download the raw .csv data by copying the data from a cloud storage bucket.

```
if not os.path.isdir("../data/transport"):
    os.makedirs("../data/transport")
```

✓ 0.0s Python

```
!gsutil cp gs://cloud-training/mlongcp/v3.0_MLonGC/toy_data/untidy_vehicle_data_toy.csv ../data/transport
```

✓ 11.9s Python

```
Copying gs://cloud-training/mlongcp/v3.0_MLonGC/toy_data/untidy_vehicle_data_toy.csv...
/ [0 files][    0.0 B/ 23.7 KiB]
/ [1 files][ 23.7 KiB/ 23.7 KiB]

Operation completed over 1 objects/23.7 KiB.
```

Read Dataset into a Pandas DataFrame

Next, let's read in the dataset just copied from the cloud storage bucket and create a Pandas DataFrame. We also add a Pandas .head() function to show you the top 5 rows of data in the DataFrame. Head() and Tail() are "best-practice" functions used to investigate datasets.

```
df_transport = pd.read_csv('../data/transport/untidy_vehicle_data_toy.csv')
df_transport.head() # Output the first five rows.
```

[10] ✓ 0.0s Python

	Date	Zip Code	Model Year	Fuel	Make	Light_Duty	Vehicles
0	10/1/2018	90000.0	2006	Gasoline	OTHER/UNK	NaN	1.0
1	10/1/2018	NaN	2014	Gasoline	NaN	Yes	1.0
2	NaN	90000.0	NaN	Gasoline	OTHER/UNK	Yes	NaN
3	10/1/2018	90000.0	2017	Gasoline	OTHER/UNK	Yes	1.0
4	10/1/2018	90000.0	<2006	Diesel and Diesel Hybrid	OTHER/UNK	No	55.0

DataFrame Column Data Types

DataFrames may have heterogenous or "mixed" data types, that is, some columns are numbers, some are strings, and some are dates etc. Because CSV files do not contain information on what data types are contained in each column, Pandas infers the data types when loading the data, e.g. if a column contains only numbers, Pandas will set that column's data type to numeric: integer or float.

Run the next cell to see information on the DataFrame.

```
df_transport.info()
```

[6] Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 499 entries, 0 to 498
Data columns (total 7 columns):
Date          497 non-null object
Zip Code      497 non-null object
Model Year    497 non-null object
Fuel          497 non-null object
Make          496 non-null object
Light_Duty    496 non-null object
Vehicles      496 non-null float64
dtypes: float64(2), object(5)
memory usage: 27.4+ KB
```

→ DataColumn Showing → String ⇒ Object

⇓

Its important to change data type format

Summary Statistics

At this point, we have only one column which contains a numerical value (e.g. Vehicles). For features which contain numerical values, we are often interested in various statistical measures relating to those values. We can use .describe() to see some summary statistics for the numeric fields in our dataframe. Note, that because we only have one numeric feature, we see only one summary stastic - for now.

```
df_transport.describe()
```

[8] Python

	Zip Code	Vehicles
count	497.00000	496.000000
mean	89838.23340	74.512097
std	3633.35609	243.839871
min	9001.00000	1.000000
25%	90001.00000	14.000000
50%	90001.00000	25.000000
75%	90001.00000	56.250000
max	90002.00000	3178.000000

describe numerical and Categorical feature

We Want to look for Something more in the Data

Diffrunt

Let's investigate a bit more of our data by using the .groupby() function.

```
[9]  df_transport.groupby('Fuel').first() # Get the first entry for each month.
                                                                          Python
```

Fuel	Date	Zip Code	Model Year	Make	Light_Duty	Vehicles
Battery Electric	10/1/2018	90000.0	<2006	OTHER/UNK	No	4.0
Diesel and Diesel Hybrid	10/1/2018	90000.0	<2006	OTHER/UNK	No	55.0
Flex-Fuel	10/14/2018	90001.0	2007	Type_A	Yes	78.0
Gasoline	10/1/2018	90000.0	2006	OTHER/UNK	Yes	1.0
Hybrid Gasoline	10/24/2018	90001.0	2009	OTHER/UNK	Yes	18.0
Natural Gas	10/25/2018	90001.0	2009	OTHER/UNK	No	2.0
Other	10/8/2018	90000.0	<2006	OTHER/UNK	Yes	6.0
Plug-in Hybrid	11/2/2018	90001.0	2012	OTHER/UNK	Yes	1.0

grouping of Data by Diffrunt Kind of fuel and we want to show first Entry for Each of the months

After this try to Clean our Data

(1) Check missing Value

Checking for Missing Values

Missing values adversely impact data quality, as they can lead the machine learning model to make inaccurate inferences about the data. Missing values can be the result of numerous factors, e.g. "bits" lost during streaming transmission, data entry, or perhaps a user forgot to fill in a field. Note that Pandas recognizes both empty cells and "NaN" types as missing values.

Let's show the null values for all features in the DataFrame.

```
    df_transport.isnull().sum()
                                                                          Python
```

```
Date          2
Zip Code      2
Model Year    2
Fuel          2
Make          3
Light_Duty    3
Vehicles      3
dtype: int64
```

```
print (df_transport['Date'])
print (df_transport['Date'].isnull())
```

① ②

Checking for Null

```
0      10/1/2018
1      10/1/2018
2            NaN
```

①

```
Name: Date, Length: 499, dtype: object
0      False
1      False
2       True
```

②

Now we need to solve this Null Values.

for doing Summerization

- Number of Example
- Num of features
- Num of unique
- which are the unique Values that we have
- For features which are missing

we use this function from Pandas

What can we deduce about the data at this point?

First, let's summarize our data by row, column, features, unique, and missing values,

```
print ("Rows     : " ,df_transport.shape[0])
print ("Columns  : " ,df_transport.shape[1])
print ("\nFeatures : \n" ,df_transport.columns.tolist())
print ("\nUnique values :  \n",df_transport.nunique())
print ("\nMissing values :  ", df_transport.isnull().sum().values.sum())
```

→ Number of rows
→ number of columns

```
Rows     :  499
Columns  :  7

Features :
 ['Date', 'Zip Code', 'Model Year', 'Fuel', 'Make', 'Light_Duty', 'Vehicles']

Unique values :
 Date          130
Zip Code         4
Model Year      15
Fuel             8
Make            43
Light_Duty       2
Vehicles       151
dtype: int64

Missing values :  17
```

Why we need to know number of unique Values in Each feature ?

↳ Let say we have days column

we have 125 unique Value

we max have 31 Days in month

This is wrong Beco2

So for this first we need to understand

Let's see the data again -- this time the last five rows in the dataset.

```
df_transport.tail()
```

5]

	Date	Zip Code	Model Year	Fuel	Make	Light_Duty	Vehicles
494	12/3/2018	90002.0	2010	Gasoline	Type_I	Yes	11.0
495	12/4/2018	90002.0	2010	Gasoline	Type_B	Yes	58.0
496	12/5/2018	90002.0	2010	Gasoline	Type_C	Yes	45.0
497	12/6/2018	90002.0	2010	Gasoline	Type_J	Yes	82.0
498	12/7/2018	90002.0	2010	Gasoline	Type_J	Yes	12.0

What Are Our Data Quality Issues?

1. **Data Quality Issue #1**:

> **Missing Values**: Each feature column has multiple missing values. In fact, we have a total of 18 missing values.

2. **Data Quality Issue #2**:

> **Date DataType**: Date is shown as an "object" datatype and should be a datetime. In addition, Date is in one column. Our business requirement is to see the Date parsed out to year, month, and day.

3. **Data Quality Issue #3**:

> **Model Year**: We are only interested in years greater than 2006, not "<2006".

4. **Data Quality Issue #4**:

> **Categorical Columns**: The feature column "Light_Duty" is categorical and has a "Yes/No" choice. We cannot feed values like this into a machine learning model. In addition, we need to "one-hot encode the remaining "string"/"object" columns.

5. **Data Quality Issue #5**:

> **Temporal Features**: How do we handle year, month, and day?

Solve the missing Values

we have 17m → Missing Values

↓

As Most Algo donot accept missing Value

↓

↳ People use to do → Drop all missing Values Rows

Tapar Bhi → Pandas fill the Rows By NAN

Few Method → To handle → Missing Value

Numeric Columns → use his Mean

Catgorical Columns → Mode

(Most frequent Value)

We can use .apply

- lambda function → for Most frequent Value in Every Column

```
df_transport.isnull().sum()
```

✓ 0.0s Python

```
Date          2
Zip Code      2
Model Year    2
Fuel          2
Make          3
Light_Duty    3
Vehicles      3
dtype: int64
```

Lab Task #1b: Apply the lambda function.

```
df_transport = df_transport.apply(lambda x:x.fillna(x.value_counts().index[0]))
```

✓ 0.0s Python

Lab Task #1c: Check again for missing values.

```
df_transport.isnull().sum()
```

✓ 0.0s Python

```
Date          0
Zip Code      0
Model Year    0
Fuel          0
Make          0
Light_Duty    0
Vehicles      0
```

df_transport.apply(lambda X: X.fillna(X.value_counts().index[0]))

↳ good for beginners approaches

→ Similar performance → idxmax() → It only access the first Element

↳ Least Versatile. → Doesn't handles → ties or return → return multiple values

less flexible

df['Vehicles'].value_count.idxmax()

for → Single frequent Value most effective and ↳ efficient → with large data set

Agar Multiple Value hai → to arbitrary Pick one

df['Vehicles'].mode().iloc[0]:

↳ ye return → all Values → with the highest frequency

↳ good hai → Multiple most frequent Values ke liye

But Performance ↓ → doing Extra process

Date quality issue →

Convert Date Column → String Datatype → to Date time format

Lab Task #2a: Convert the datetime datatype with the to_datetime() function in Pandas.

```
df_transport['Date']= pd.to_datetime(df_transport['Date'], format='%m/%d/%Y')
```

✓ 0.0s

Lab Task #2b: Show the converted Date.

```
df_transport.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 499 entries, 0 to 498
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Date        499 non-null    datetime64[ns]
 1   Zip Code    499 non-null    float64
 2   Model Year  499 non-null    object
 3   Fuel        499 non-null    object
```

Method	Pros	Cons
`pd.to_datetime(format='')`	Concise, efficient, works for standard formats	Limited flexibility for non-standard formats
`datetime.datetime.strptime`	Highly flexible, handles various formats	Requires more code, error-prone for complex formats
`pd.to_datetime(infer_datetime_format=True)`	Convenient, automatic detection	Might not always be accurate, not ideal for mixed formats
`pd.coerce_datetime`	Robust, handles mixed formats and missing values	Might not convert to desired format, less control

Here we are Breaking Date time into month year Date

Let's parse Date into three columns, e.g. year, month, and day.

```
df_transport['year'] = df_transport['Date'].dt.year
df_transport['month'] = df_transport['Date'].dt.month
df_transport['day'] = df_transport['Date'].dt.day
#df['hour'] = df['date'].dt.hour - you could use this if your date format included hour.
#df['minute'] = df['date'].dt.minute - you could use this if your date format included minute.
df_transport.info()
```

Pytho

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 499 entries, 0 to 498
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         499 non-null    datetime64[ns]
 1   Zip Code     499 non-null    float64
 2   Model Year   499 non-null    object
 3   Fuel         499 non-null    object
 4   Make         499 non-null    object
 5   Light_Duty   499 non-null    object
 6   Vehicles     499 non-null    float64
 7   year         499 non-null    int64
 8   month        499 non-null    int64
 9   day          499 non-null    int64
dtypes: datetime64[ns](1), float64(2), int64(3), object(4)
memory usage: 39.1+ KB
```

Next, let's confirm the Date parsing. This will also give us a another visualization of the data.

```
# Here, we are creating a new dataframe called "grouped_data" and grouping by on the column "Make"
grouped_data = df_transport.groupby(['Make'])

# Get the first entry for each month.
df_transport.groupby('month').first()
```

Python

month	Date	Zip Code	Model Year	Fuel	Make	Light_Duty	Vehicles	year	day
1	2019-01-01	90001.0	2016	Gasoline	Type_G	Yes	18.0	2019	1
2	2019-02-01	90001.0	2017	Gasoline	Type_D	Yes	13.0	2019	1
3	2019-03-01	90001.0	2018	Gasoline	Type_C	Yes	32.0	2019	1
10	2018-10-01	90000.0	2006	Gasoline	OTHER/UNK	Yes	1.0	2018	1
11	2018-11-01	90001.0	2007	Gasoline	Type_M	Yes	15.0	2018	1
12	2018-12-02	90001.0	2015	Gasoline	Type_G	Yes	19.0	2018	2

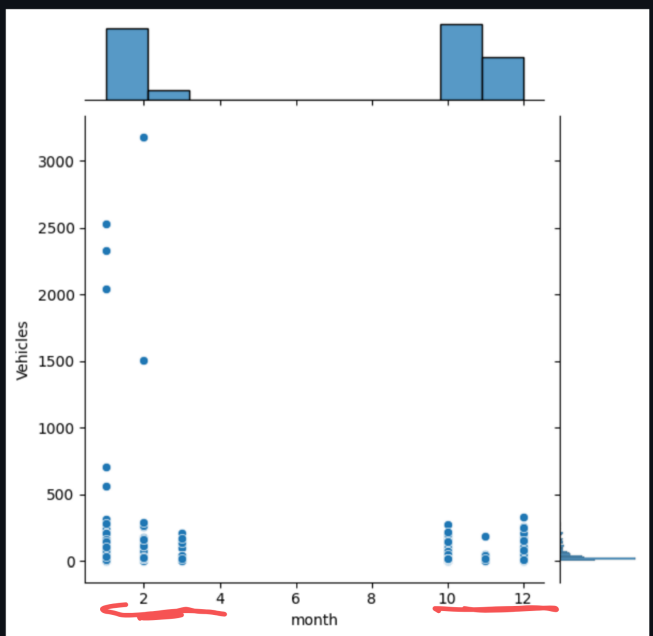
Now that we have Dates as a integers, let's do some additional plotting.

```
plt.figure(figsize=(10,6))
sns.jointplot(x='month',y='Vehicles',data=df_transport)
#plt.title('Vehicles by Month')
```

4] ✓ 1.2s

```
<seaborn.axisgrid.JointGrid at 0x176d3d30730>

<Figure size 1000x600 with 0 Axes>
```



here we have data only few months

Data quality :→ Rename a feature Column and Remove Value

```
df_transport.rename(columns = {col : col.lower().replace(' ','') for col in df_transport.columns}, inplace = True)
```

inplace → Whether changes need to apply in original Dataframe (True)

or
a copy (false)

Method	Description	Pros	Cons
String methods	`df.columns = df.columns.str.lower().str.replace(' ', '_')`	Simple and concise, efficient for small datasets	Less readable with multiple methods, may not handle special characters well
List comprehension	`df.columns = [col.lower().replace(' ', '_') for col in df.columns]`	More explicit and easier to understand, easily extensible	Slightly less concise, slightly less efficient for large datasets due to loop
pd.DataFrame.rename	`df.rename(columns={col: col.lower().replace(' ', '_') for col in df.columns}, inplace=True)`	Most versatile, handles complex mappings and logic, supports in-place modification	More verbose, overkill for simple renaming
Factors to consider:			
- Size and complexity of data	Larger datasets might favor efficiency.		
- Readability preference	Choose an approach you find clear and maintainable.		
- Potential future modifications	Consider future renaming needs and flexibility.		

```
df_transport.rename(columns={col: col.lower().replace(' ','')for col in df_transport.columns}, inplace=True)
df_transport.head(2)
```

✓ 0.0s

	date	zipcode	modelyear	fuel	make	light_duty	vehicles	year	month	day
0	2018-10-01	90000.0	2006	Gasoline	OTHER/UNK	Yes	1.0	2018	10	1
1	2018-10-01	90001.0	2014	Gasoline	OTHER/UNK	Yes	1.0	2018	10	1

Fuel						
Battery Electric	10/1/2018	90000.0	<2007	OTHER/UNK	No	4.0
Diesel and Diesel Hybrid	10/1/2018	90000.0	<2006	OTHER/UNK	No	55.0
Flex-Fuel	10/14/2018	90001.0	2007	Type_A	Yes	78.0

we Cant Process this Kind of Data

we need Proper form this

Lab Task #3b: Create a copy of the dataframe to avoid copy warning issues.

```
df = df_transport.loc[df_transport.modelyear != '<2006'].copy()
```

✓ 0.0s

Next, confirm that the modelyear value '<2006' has been removed by doing a value count.

```
df['modelyear'].value_counts(0)
```

✓ 0.0s

```
modelyear
2007    53
2008    45
2006    36
2010    34
2014    31
```

So we need to transform this modul year

we eliminated all

↳ If we work → with year / model_year

model_year → as Categorical Column

So we ← Can have ← Out of Vocabulary list

All will be inside ←

But here we eliminated

Data quality → Handling Categorical Columns

↓

Light Duty → [yes] or [no]

we Can't ↙ find Value like this in

ML
DL
AI

we have to ← Convert it into [0] [1]

Various Method → to do this we will use

Pandas.apply( )

↳ In this Lambda function

What is Lambda function ?

Python → mai → need hai ki function define no → def keyword se

But Lambda function → anonymous hai

which means hum is no need to name them ←

First, lets count the number of "Yes" and"No's" in the 'lightduty' feature column.

```
df['light_duty'].value_counts(0)
```

✓ 0.0s Python

```
light_duty
Yes    374
No      42
Name: count, dtype: int64
```

Let's convert the Yes to 1 and No to 0. Pandas. apply() . apply takes a function and applies it to all values of a Pandas series (e.g. lightduty).

```
df.loc[:,'light_duty'] = df['light_duty'].apply(lambda x: 0 if x=='No' else 1)
df['light_duty'].value_counts(0)
```

✓ 0.0s Python

```
light_duty
1    374
0     42
Name: count, dtype: int64
```

```
# Confirm that "lightduty" has been converted.

df.head()
```

✓ 0.0s Python

	date	zipcode	modelyear	fuel	make	light_duty	vehicles	year	month	day
0	2018-10-01	90000.0	2006	Gasoline	OTHER/UNK	1	1.0	2018	10	1
	2018-10-									

One - hot coding

We will do Same to all → Categorical Column

ML → To hai → wo Expect Karta hai → input Vector

(not Categorical features) ← they Cannot handle text

↳ or → String Values

Here we Can do → Lambda function → for One-hot Encoding

↳ or we Can do → get_dummies

↓

It will convert Categorical Variable to To numerical Value

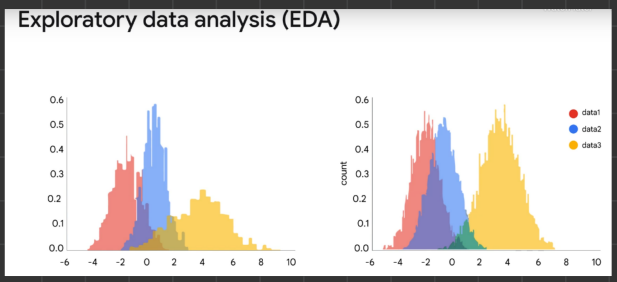
Exploratory Data Analysis

↳ Approach → which is for → Analyzing data set to Summarization ↙ the main characteristic

EDA → helps what [Data] → Can tells Beyond ↓ the formal Moldeling or
- hypothesis testing task

EDA → for graphics
↳ Basic stastics such → As
- Mean
- midian
- Standard deviation

- look at data of trends, outlier and patterns



→ These all thing Can be tested in modeling steps

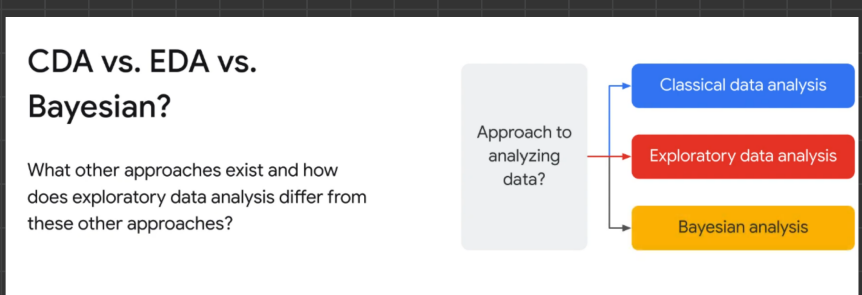
EDA Techniques →

- ↳ Mostly graphical
- ↳ Maximize insight into a Dataset
- ↳ Uncover Underlying Structure
- ↳ Extract important Variable

- ↳ detect outlier
- ↳ Anomalies
- ↳ test Underlying assumption
- ↳ develop
  - ↳ Parsimonial models
- ↳ determine optimal factory setting

(3) → Data analysis Aproch are

- EDA
- Bayesian
- Classical



↓

3 approaches are Similar Only

↓

Difference is the sequence and focus

Intermediate Steps

Problem ⇒ Data ⇒ Model ⇒ Analysis ⇒ Conclusion

Classical → follows as predetermined Model

- Normality ← like
- linearity

Focus on

↳ Estimating } → Parameter with in the Model.
↳ testing }

Problem ⇒ Data ⇒ Analysis ⇒ Model ⇒ Conclusion

EDA → Starts By analyzing the Data

Without assumption about its structure

Focus is

Put the appropriate model

Based on the Data Analysis

↳ Allow Data → to Suggest → Best fitting model

Problem ⇒ Data ⇒ Modul ⇒ Prior Distribution

Baysian :⇒ Analysis = Conclusion

focus → on Research Kind of thing

↓

finding thing about unknown Paramtor

using Probability Statements

EdA → doesnt impose deterministic → Probabilistic → modul on Data

↓

(ike Classical

Purpose of Baysian Analysis

↓

is to determine Posterior Probabilities based on Prior Probabilities

Posterior Prob → Data on background information

↳ Evidence

will be taken after Event

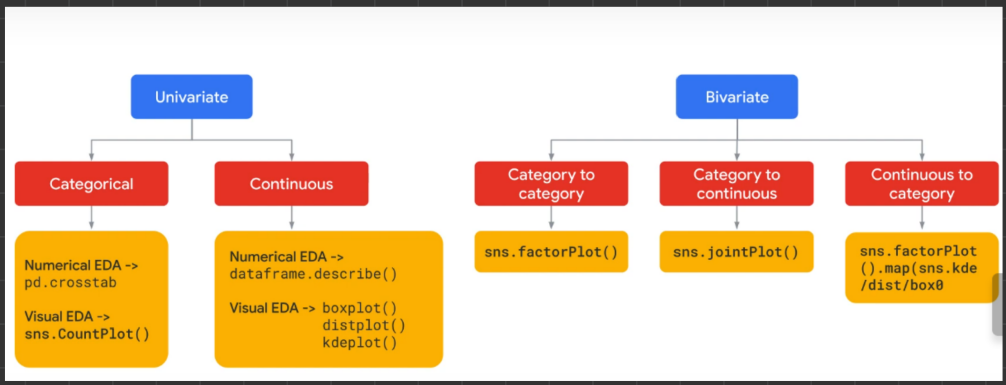
Prior Probabilities → Befor Event

How EDA used in ML ?

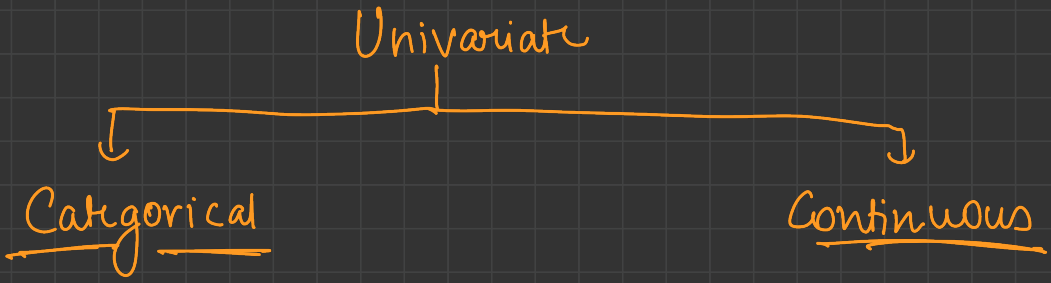
EDA → Allows → Data → to Suggest a model for themself.

So need to focus on Data

- Structure
- Outlier
- Models suggest by Data



Exploratory Data Analysis (EDA) typically Performed Using above Methods


Univariate
Categorical
Continuous