PYTHON FOR DATA SCIENCE

INTRODUCTION TO PROGRAMMING

About Me

My name is Daniel Bissell.

- I started my career in neuroscience research, exploring the intricacies of the human brain and behavior.
- Transitioned to cybersecurity and protecting digital assets and combating cyber threats.
- Though different in many ways, both fields require attention to detail, critical thinking, and a drive to solve complex problems.
- And Python!

WHAT IS PYTHON?

PYTHON

- PYTHON IS A HIGH-LEVEL, INTERPRETED PROGRAMMING LANGUAGE WIDELY USED IN DATA SCIENCE AND MACHINE LEARNING.

- PYTHON HAS A SIMPLE SYNTAX AND A VAST LIBRARY OF MODULES, MAKING IT A POPULAR CHOICE FOR DATA ANALYSIS AND VISUALIZATION.

- SOME POPULAR LIBRARIES FOR DATA SCIENCE IN PYTHON INCLUDE NUMPY, PANDAS, MATPLOTLIB, AND SCIKIT-LEARN.

Course Objectives

Data Types

Different kinds of values that can be stored and manipulated by a program.

Data Structures

A way of organizing and storing data in a program.

Conditions

Control the flow of a program based on certain criteria.

Functions

Reusable blocks of code that perform a specific task.

Loops

Repeat a block of code multiple times.

Python

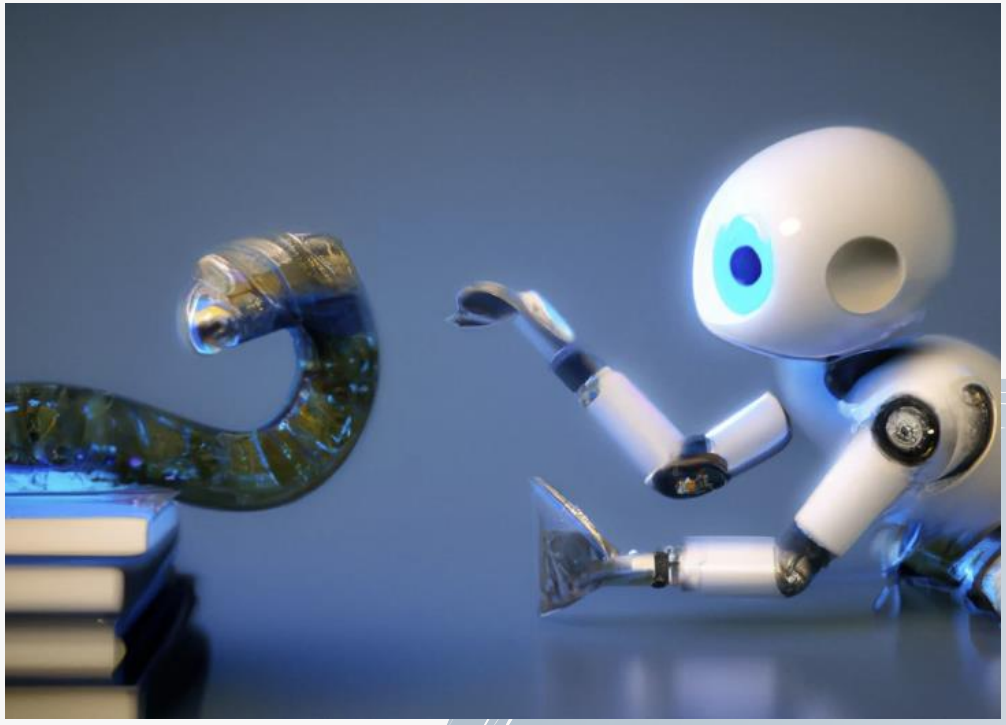
- Python is an interpreter language.
- Python is an Object-Oriented language
- Performs instruction validation
- Results are returned in sequential order.

IDEs

Most python is used with an IDE

Python can be ran without these, just as a .py file

Object-Oriented



- Object-oriented programming (OOP): Object-oriented programming is a programming paradigm that involves organizing code into objects, which can interact with each other to perform tasks. OOP is based on the concepts of classes and objects.
- Objects are data types that contain attributes and functionalities
- Classes: In OOP, a class is a blueprint for creating objects. A class defines the properties and methods that an object will have.

Python Syntax

- Hashtags/Pound (#) will not execute code on that line, often used to write comments in code
- 3 quotes are often used for multiline comments ''' '''
- Statements must be correctly written(control statements musts contain a colon)

```
print('This is python code')

# This is a python comment
```

DATA TYPES

Variables

```
name = 'alice'
name2 = 'bob'

x = 3
y = 6

xf = 3.6
yf = 6.3

statement = True
statement2 = False
```

- Variables are names that hold values. (Think algebra x=5)
- In python variables are used store data. Assign with =
- Python can automatically detect the data type
- Variable names must be with an underscore or letter

Data Types

```
name = 'alice'
name2 = 'bob'

x = 3
y = 6

xf = 3.6
yf = 6.3

statement = True
statement2 = False
```

- Strings
- Ints
- Floats
- Boolean

Types - Detect and Cast

```
name = 'alice'
name2 = 'bob'

x = 3
y = 6

xf = 3.6
yf = 6.3

statement = True
statement2 = False
```

```
print(type(name))
print(type(x))
print(type(xf))
print(type(statement))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

- To change the type(cast)

Mathematical Operations

```
In [55]: z = x + y; print(z)
9
```

```
In [56]: z = x - y; print(z)
-3
```

- + Addition
- - Subtraction
- * Multiplication
- ** Exponent
- / Division
- // Integer divison
- % Modulo

PRINTING DATA

Printing Data

- print(x)
- print(f"The value of {x} is")
- print("The value of "+str(x)+" is")

In order to interact with out variables we will introduce a function, print. Print will print data to our console allowing us to read it

```
x = 6
print(x)
print(f'The value of x is {x}')
```

```
6
The value of x is 6
```

Data Type - String

```
x = 'This is a string in python!'
#Print string
print(x)
#Print slice of string
print(x[0:5])
# Print reversed string
print(x[::-1])
#Print last character in string
print(x[-1])
#Print every other
print(x[::2])
```

- Strings are ordered sequences. We can slice them.
- Upper()
- Lower()
- Split()

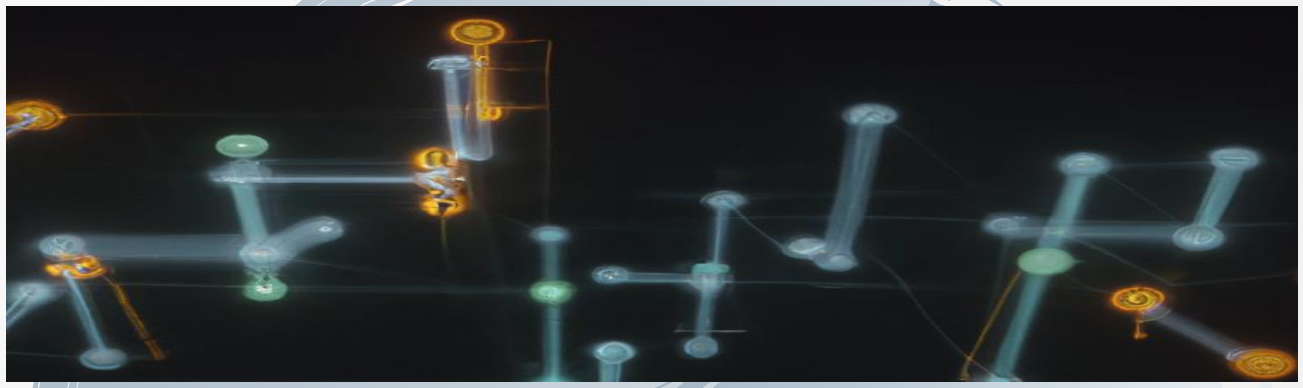
DATA STRUCTURES

Data Structures

Ways to store groups of variables. Able to detect by the brackets used to enclose the data.

i.e (,{,[...

- List
- Set
- Tuple
- Dictionary



Data Structures - List

Can slice and manipulate lists like strings

- Lists: In Python, a list is an ordered collection of items.
- Lists are mutable, which means you can add, remove, and modify items in a list.
- Lists are created using square brackets and items are separated by commas.

```
x = ['This', 'is', 'a', 'list', 'in', 'python!']
print(x)
```

Data Structures - Set

```
# Create two sets
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}

# Find their intersection
intersection = set1.intersection(set2)

# Print the result
print("The intersection of set1 and set2 is:", intersection)
```

```
The intersection of set1 and set2 is: {4, 5}
```

- Sets: In Python, a set is an unordered collection of unique elements.
- Sets are mutable, which means you can add and remove elements from a set.
- Sets are created using curly braces or the `set()` function.

Data Structures - Tuple

```
# Create two tuples

x = (2,4)
y = (3,6)
```

- Tuples: In Python, a tuple is an ordered, immutable collection of elements.
- Tuples are similar to lists, but once you create a tuple, you cannot modify its contents.
- Tuples are created using parentheses and items are separated by commas.

Data Structures - Dictionary

```
# Create a dictionary
my_dict = {'apple': 1, 'banana': 2, 'orange': 3}

# Add a new entry
my_dict['pear'] = 4

# Call a value from a key
print("The value of 'banana' is:", my_dict['banana'])
```

```
The value of 'banana' is: 2
```

- Dictionaries: In Python, a dictionary is an unordered collection of key-value pairs.
- Dictionaries are mutable, which means you can add, remove, and modify key-value pairs in a dictionary.
- Dictionaries are created using curly braces and key-value pairs are separated by colons.

CONDITIONS

Conditionals

```
# Define variables x and y
x = 3
y = 6
z = 5


# Check if x is equal to y
if x == y:
    print("x is equal to y")

# Check if x is not equal to y
if x != y:
    print("x is not equal to y")



# Check if x is greater than y
if x > y:
    print("x is greater than y")
# Check if x is less than y
elif x < y:
    print("x is less than y")
# If neither condition is true, then x must be equal to y
else:
    print("x is equal to y")



# Check if x OR y is greater than z
if x > z or y > z:
    print('x or y is bigger than z')
```

- IF
- ELSE
- OR
- NOT
- AND

Comparisons

```
# Define variables x and y
x = 3
y = 6
z = 5


# Check if x is equal to y
if x == y:
    print("x is equal to y")

# Check if x is not equal to y
if x != y:
    print("x is not equal to y")



# Check if x is greater than y
if x > y:
    print("x is greater than y")
# Check if x is less than y
elif x < y:
    print("x is less than y")
# If neither condition is true, then x must be equal to y
else:
    print("x is equal to y")



# Check if x OR y is greater than z
if x > z or y > z:
    print('x or y is bigger than z')
```

- >
- <
- <=
- >=
- ==
- !=

FUNCTIONS

Functions

- We have used functions already, print.
- Allows for input and output
- With data we are often going to want to manipulate it in some way.

Functions - Declaration

- Start with def
- Function name
- () with input
- : to end

```
# Declare a simple function
def greet(name):
    print(f"Hello, {name}!")

# Invoke a function
greet('Bob')
```

Functions - Invocation

```
# Declare a simple function
def greet(name):
    print(f"Hello, {name}!")

# Invoke a function
greet('Bob')
```

- Defining a function and calling it are 2 different things.
- Function name
- () with input
- : to end

```
Hello, Bob!
```

Functions – Returning Values

```python
# Define a function to calculate area and perimeter of a rectangle
def rectangle_calculations(width, height):
    area = width * height
    perimeter = 2 * (width + height)
    return area, perimeter

# Call the function and get the returned values
a, p = rectangle_calculations(5, 10)
print(f"Area: {a}, Perimeter: {p}")
```

```
Area: 50, Perimeter: 30
```

- Defining a function and calling it are 2 different things.
- Function name
- () with input
- : to end

Functions – Scope

- Local variables are those that are defined within a function and are only accessible within that function's scope.
- Global variables are those that are defined outside of any function and are accessible throughout the entire program
- If you want to modify the value of a global variable within a function, you need to use the `global` keyword to indicate that you are accessing the global variable.

LOOPS

Loops

Loops are the heart of a program. Loops are block of code that run while a condition is true.

2 Types

- For loops
- While Loops

For Loops

For loops are useful to iterate through a list or container. Often used with a range function

Demo

```
# Loop over a list
my_list = [1, 2, 3, 4, 5]
for item in my_list:
    print(item)

# Loop using range
for i in range(1, 6):
    print(i)
```

For Loops

```
# Enumerate a list
my_list = ['apple', 'banana', 'orange']
for index, item in enumerate(my_list):
    print(index, item)
```

```
0 apple
1 banana
2 orange
```

Enumerate

- The enumerate function is a built-in Python function that allows us to iterate over a sequence of items while keeping track of the index of each item.
- It returns a tuple for each iteration that contains the index and the corresponding item value.
- The enumerate function is commonly used in for loops when we need to access both the index and value of each item in the sequence. It is a simple and convenient way to add indexing functionality to our loops.

For Loops

```
# Create a list using list comprehension
my_list = [x**2 for x in range(1, 6)]
print(my_list)
```

```
[1, 4, 9, 16, 25]
```

List comprehension

- List comprehension is a concise and elegant way to create a new list from an existing list or other iterable in Python. It allows us to define a list using a single line of code, instead of using a for loop to append items one by one. List comprehension is a powerful feature in Python that can help simplify and optimize code.

While Loops

While loops are useful to have a chuck of code continue until a condition is met. Often used in conjunction with a Counter.

Must be careful of infinite loops

```
# Use a while loop with a counter variable
counter = 0
while counter < 5:
    print(counter)
    counter += 1
```

PUTTING IT ALL TOGETHER

FLOW CONTROL

Flow Control

```python
# Use a for loop with break, pass, and continue statements
my_list = [1, 2, 3, 4, 5]

# Iterate over the list
for item in my_list:
    # Skip even numbers
    if item % 2 == 0:
        continue
    # Print odd numbers
    print(item)
    # Stop after reaching 3
    if item == 3:
        break
    # Placeholder for future code
    pass
```

- Break
- Continue
- Pass

ERROR HANDLING

Error Handling

```
During handling of the above exception, another exception occurred:

Traceback (most recent call last):

  File ~\.spyder-py3\untitled4.py:16 in <module>
    print('error with'+entry)

TypeError: can only concatenate str (not "tuple") to str
```

- The `try` block contains the code that you want to monitor for exceptions.
- If an exception occurs within the `try` block, the program execution is transferred to the corresponding `except` block.

Error Handling

```python
def divide_numbers(a, b):
    try:
        result = a / b
    except ZeroDivisionError:
        print("Error: Cannot divide by zero.")
        return None
    except TypeError:
        print("Error: Please provide valid numeric inputs.")
        return None
    else:
        return result
```

```python
# Test cases
print(divide_numbers(10, 2))
print(divide_numbers(5, 0))
print(divide_numbers("a", 2))
```

```
5.0
Error: Cannot divide by zero.
None
Error: Please provide valid numeric inputs.
None
```

CLASS CREATION

Class Creation

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Creating objects of class Person
person1 = Person("Alice", 30)
person2 = Person("Bob", 25)

# Accessing object attributes
print(person1.name)    # Output: Alice
print(person2.age)     # Output: 25

# Calling object method
person1.greet()   # Output: Hello, my name is Alice and I am 30 years old.
person2.greet()   # Output: Hello, my name is Bob and I am 25 years old.
```

```
In [166]: runfile('C:/Users/Daniel/.spyder-py3/untitled4.py', wdir='C:/Users/Daniel/.spyder-py3')
Alice
25
Hello, my name is Alice and I am 30 years old.
Hello, my name is Bob and I am 25 years old.
```

- Greet method

INTERACTING WITH FILE SYSTEM

Course Review

Data Types

Different kinds of values that can be stored and manipulated by a program.

Data Structures

A way of organizing and storing data in a program

Conditions

Control the flow of a program based on certain criteria.

Functions

Reusable blocks of code that perform a specific task.

Loops

Repeat a block of code multiple times.

Extra Resources

Python Practice Problems

- Hackerrank.com
- Edabit.com
- Leetcode.com

PYTHON FOR DATA SCIENCE

- High level programming language
- Built to be human readable
- Includes many libraries or modules

Extra Resources

- https://pandas.pydata.org/docs/reference/io.html
- https://scikit-learn.org/stable/
- https://seaborn.pydata.org/
- https://jakevdp.github.io/PythonDataScienceHandbook/

Data Science Overview

Data Exploration

What do I have?

What is it missing?

Pre-Processing

Clean Data

Feature Engenieering

Data Analysis

Identify Patterns

Visualize Data

Model Building

Select Algorithm

Prepare Data

Model Deployment

Train/Test

Fine Tune

INTRODUCTION TO PANDAS

HTTPS://WWW.KAGGLE.COM/DATASETS/NEHALBIRLA/VEHICLE-DATASET-FROM-CARDEKHO?SELECT=CAR+DATA.CSV

Pandas – Load a CSV

```
my_df = pd.read_csv('C:\\Users\\Daniel\\Downloads\\archive\\car data.csv')
```

Index	Car Name	Year	Selling Price	Present Price	ms Drive	Fuel Type	Seller Type
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer
4	swift	2014	4.6	6.87	42450	Diesel	Dealer
5	vitara brezza	2018	9.25	9.83	2071	Diesel	Dealer

Pandas Data Types

```
import pandas as pd
import numpy as np

# Creating a DataFrame with each of the requested data types
df = pd.DataFrame({
    'Object_Column': ['Text', 'More Text', 'Another Text', 'Final Text'],
    'Int_Column': [1, 2, 3, 4],
    'Float_Column': [1.1, 2.2, 3.3, 4.4],
    'Bool_Column': [True, False, True, False]
})

print(df.dtypes)
print('')
print(df.Float_Column.dtype)
```

```
In [5]: runcell(9, 'C:/Users/Daniel/.spyder-py3/python_data_science_class_notes.py')
Object_Column     object
Int_Column         int64
Float_Column     float64
Bool_Column         bool
dtype: object

float64
```

Pandas Data Structures

2 Types

- Series
- DataFrame

```
import pandas as pd

my_series = pd.Series([1, 2, 3, 4, 5])

my_df = pd.DataFrame({'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]})
```

Index	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	35

Pandas – DataFrame

```
count     301.000000
mean     2013.627907
std         2.891554
min      2003.000000
25%      2012.000000
50%      2014.000000
75%      2016.000000
max      2018.000000
Name: Year, dtype: float64
count        301
unique         3
top       Petrol
freq         239
Name: Fuel_Type, dtype: object
```

Info/Describe

- my_df.info()
- my_df.describe()
- my_df.columns

- Can run stats on a single column
- my_df.Year.describe()
- my_df['Year'].describe()
- (Both are same, 2 ways to reference a column)
- Describe is aware of the datatype i.e object vs float.

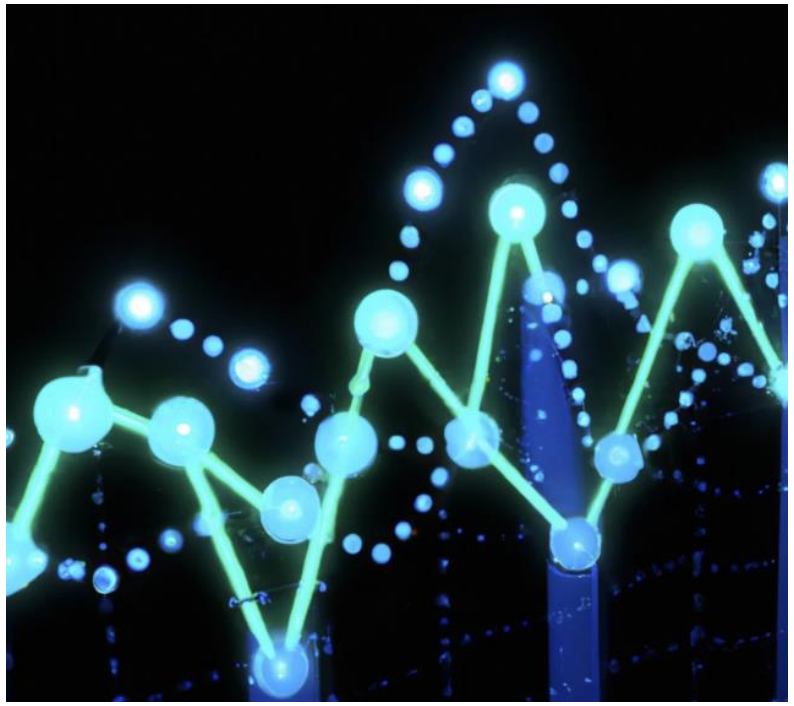
Pandas – DataFrame

```
count     301.000000
mean     2013.627907
std         2.891554
min      2003.000000
25%      2012.000000
50%      2014.000000
75%      2016.000000
max      2018.000000
Name: Year, dtype: float64
count        301
unique         3
top       Petrol
freq         239
Name: Fuel_Type, dtype: object
```

Summary Stats

- df.Object_Column.unique()
- df.Object_Column.value_counts()
- df.Float_Column.mean()

Pandas – DataFrame



Missing Data

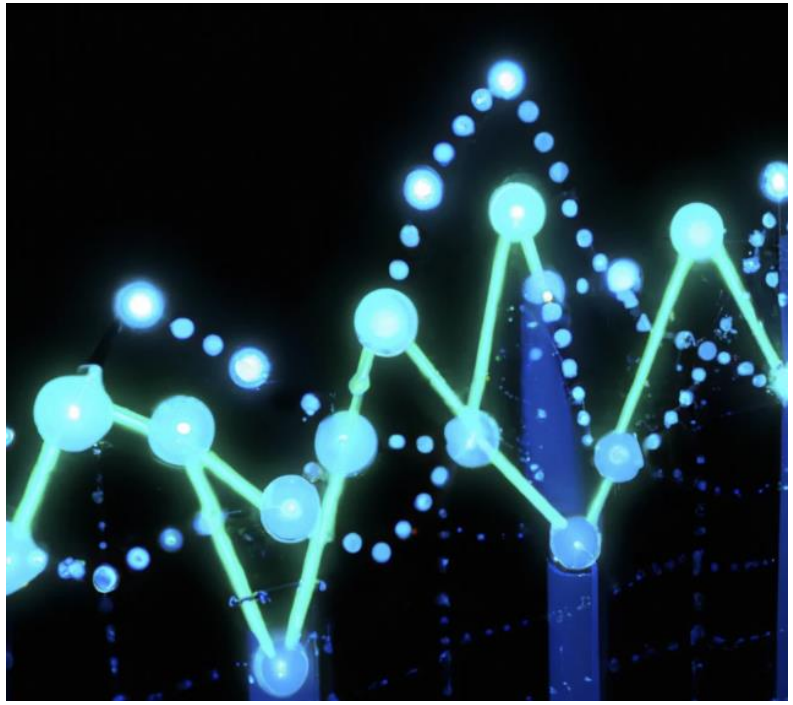
- my_df.isnull().sum()

- Drop rows if value is missing from subset column
  - my_df.dropna(subset=['Present_Price'], inplace=True)

- Drop rows with 3 or more missing values(if 10 columns would need 7 to contain values to not be dropped)
  - my_df.dropna(thresh=3, inplace=True)

- Drop rows where column 'Kms_Driven' is greater than the threshold
  - my_df = my_df[my_df['Kms_Driven'] <= 100000]

Pandas – DataFrame



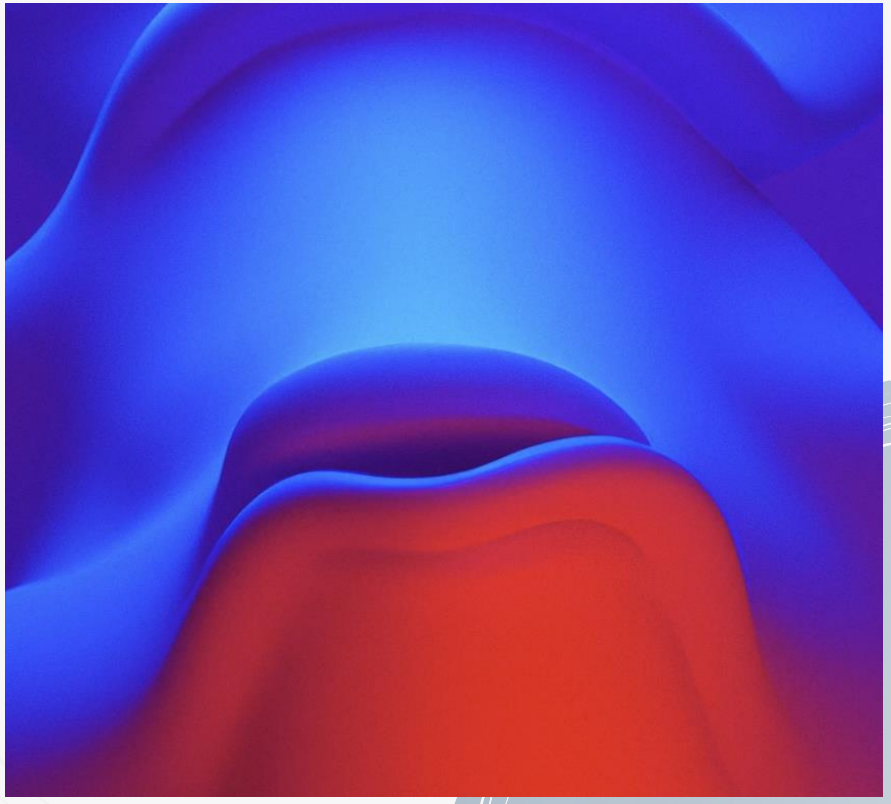
Replace Data

```
# Replace data

df.Object_Column.replace('Final Text','Really, I am done now', inplace=True)
```

df - DataFrame

Index	Object Column	nt Colum	oat Colur	ool Colun
0	Text	1	1.1	True
1	More Text	2	2.2	False
2	Another Text	3	3.3	True
3	Really, I am done now	4	4.4	False

Pandas – DataFrame



Merging

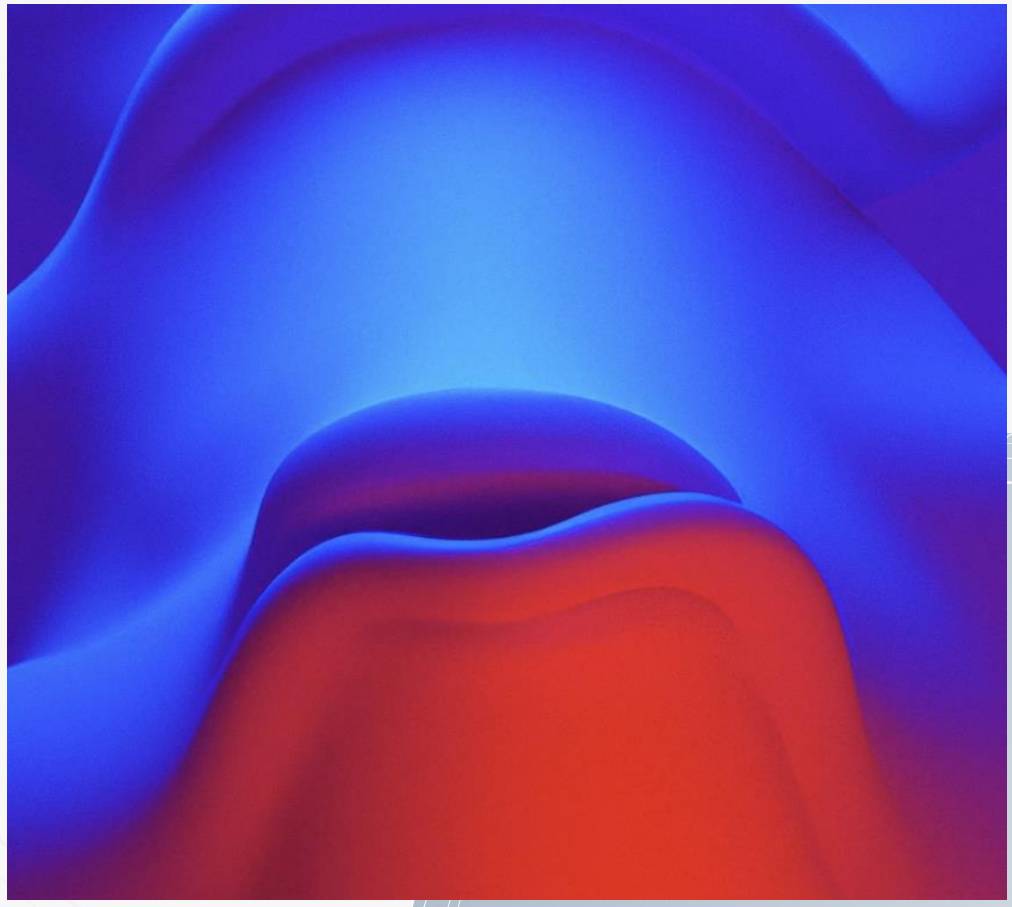
- Index

```
df_2 = my_df[['Year','Selling_Price']]
df_3 = my_df[['Kms_Driven','Transmission']]
# Merge the DataFrames on their index
merged_df = df_2.merge(df_3, left_index=True, right_index=True)
```

- Column

```
# Merge the DataFrames on the 'ID' column
merged_df = df1.merge(df2, on='ID')
```

Pandas – DataFrame



Merging

df_2 - DataFrame

Index	Year	ellinq Pric
0	2014	3.35
1	2013	4.75
2	2017	7.25
3	2011	2.85
4	2014	4.6

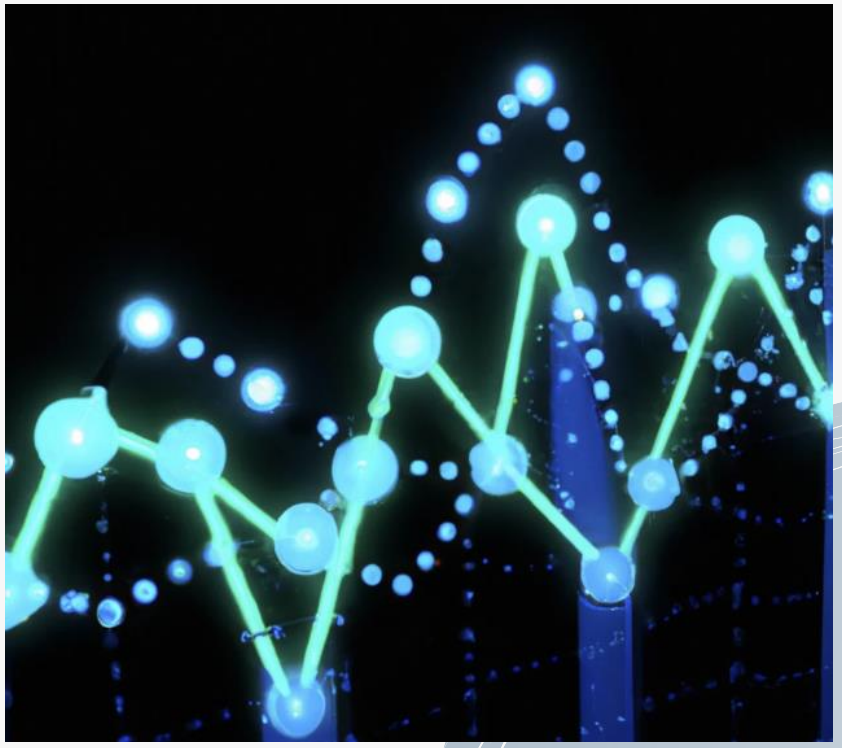
df_3 - DataFrame

Index	ms Drive	ransmissior
0	27000	Manual
1	43000	Manual
2	6900	Manual
3	5200	Manual
4	42450	Manual

merged_df - DataFrame

Index	Year	ellinq Pric	ms Drive	ransmissior
0	2014	3.35	27000	Manual
1	2013	4.75	43000	Manual
2	2017	7.25	6900	Manual
3	2011	2.85	5200	Manual
4	2014	4.6	42450	Manual

Pandas – DataFrame



Data Normalization and Scaling

- Normalization brings all features to a common scale, ensuring that no feature's influence is disproportionately amplified or diminished.
- Algorithms converge faster and perform better when features are normalized.

Pandas – DataFrame

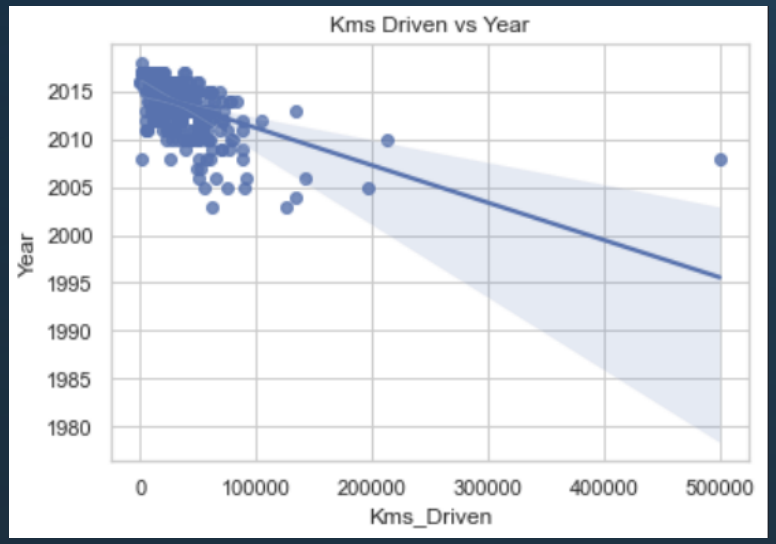
Data Normalization and Scaling

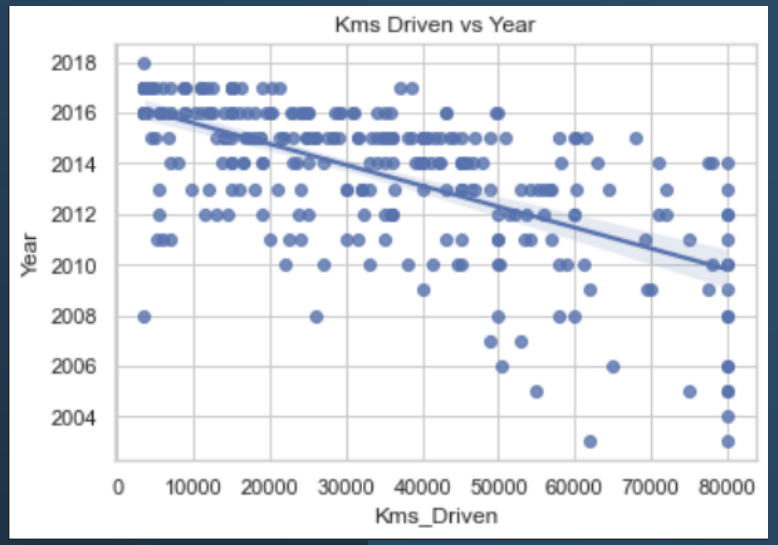
Index	Car Name	Year	ellinq Pric	esent Pri	ms Drive
0	ritz	2014	3.35	5.59	27000
1	sx4	2013	4.75	9.54	43000
2	ciaz	2017	7.25	9.85	6900

```python
from sklearn.preprocessing import MinMaxScaler

# Select columns for normalization and scaling
columns_to_normalize = ['Year', 'Selling_Price', 'Present_Price', 'Kms_Driven']

# Create a MinMaxScaler instance
scaler = MinMaxScaler()

# Fit and transform the selected columns
my_df[columns_to_normalize] = scaler.fit_transform(my_df[columns_to_normalize])

# Display the updated DataFrame
print(my_df)
```

Index	Car Name	Year	Selling Price	Present Price	Kms Driven
0	ritz	0.733333	0.0931232	0.0571088	0.0530531
1	sx4	0.666667	0.133238	0.0999133	0.0850851
2	ciaz	0.933333	0.204871	0.103273	0.0128128

Handling Outliers






```
# Handling Outliers

# Calculate the 95th percentile (quantile) of 'Kms_Driven' to detect outliers
quantile_95 = my_df['Kms_Driven'].quantile(0.95)

# Clip 'Kms_Driven' values to be within the 5th and 95th percentile range
my_df['Kms_Driven'] = my_df['Kms_Driven'].clip(lower=my_df['Kms_Driven'].quantile(0.05), upper=quantile_95)
```

Pandas – DataFrame

.at / .loc

```
#loc for multiple
# Access a specific value using row and column labels
value = my_df.loc[row_label, col_label]

# Modify multiple values at once
my_df.loc[row_label, col_label] = new_value

# Select specific rows and columns
subset = my_df.loc[row_labels, col_labels]

# Apply conditional filtering
filtered_data = my_df.loc[my_df['column_name'] > 100]

# at for single
# Access a single value using row and column labels
value = my_df.at[row_label, col_label]

# Modify a single value
my_df.at[row_label, col_label] = new_value
```

Pandas – DataFrame

Slice DataFrame

- my_df.head()
- my_df.tail()
- my_df.sample(n=5)

Pandas – DataFrame

Filtering

- filtered_df_1 = my_df.loc[my_df['Transmission'] == 'Manual']
- filtered_df_2 = filtered_df_1.loc[filtered_df_1['Kms_Driven'] < 15000]

- my_df_no_fuel = my_df.drop('Fuel_Type', axis=1)

Pandas – DataFrame Grouping

grouping

- my_df.groupby('Car_Name')['Selling_Price'].mean()

Car_Name	Selling_Price
land cruiser	35
fortuner	18.6855
innova	12.7778
creta	11.8
elantra	11.6

- my_df.groupby('Transmission')['Selling_Price'].mean()

ransmiss	Selling_Pri
Automatic	9.42
Manual	3.93199

Pandas – Correlation

```
# Print correlation
corr = my_df['Year'].corr(my_df['Kms_Driven'])
print(f"Correlation between Year and Kms_Driven: {corr}")
```

```
In [148]: runcell(7, 'C:/Users/Daniel/.spyder-py3/python_data_science_class_notes.py')
Correlation between Year and Kms_Driven: -0.5243420406957319
```

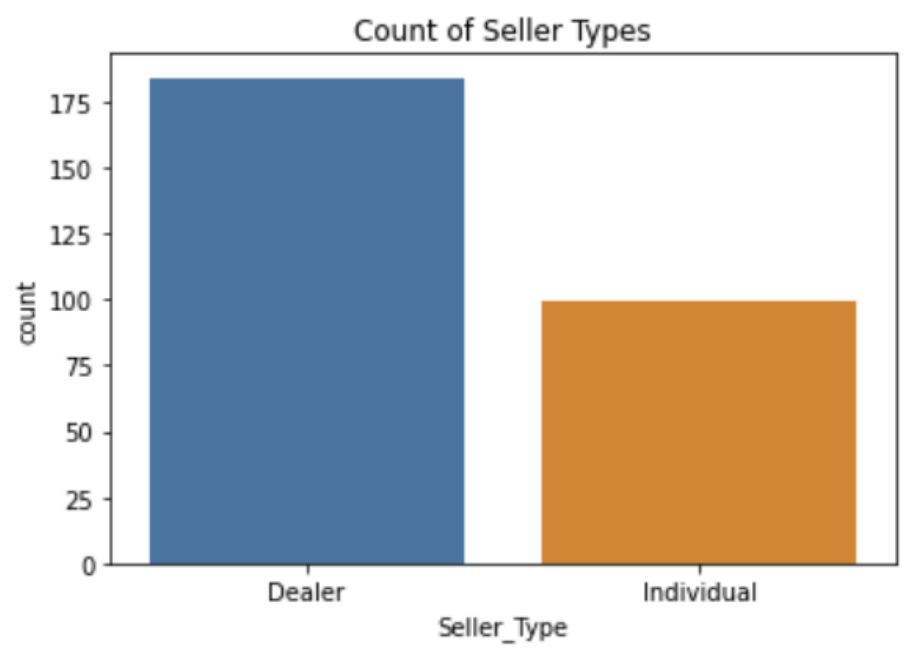
VISUALIZING DATA

Visualizing Data

Matplotlib

- Matplotlib is a popular data visualization library for Python.
- It provides a wide range of tools for creating high-quality plots, graphs, and charts, making it a valuable tool for data analysis and scientific research.
- Matplotlib is highly customizable, allowing users to create a wide range of visualizations to suit their needs.

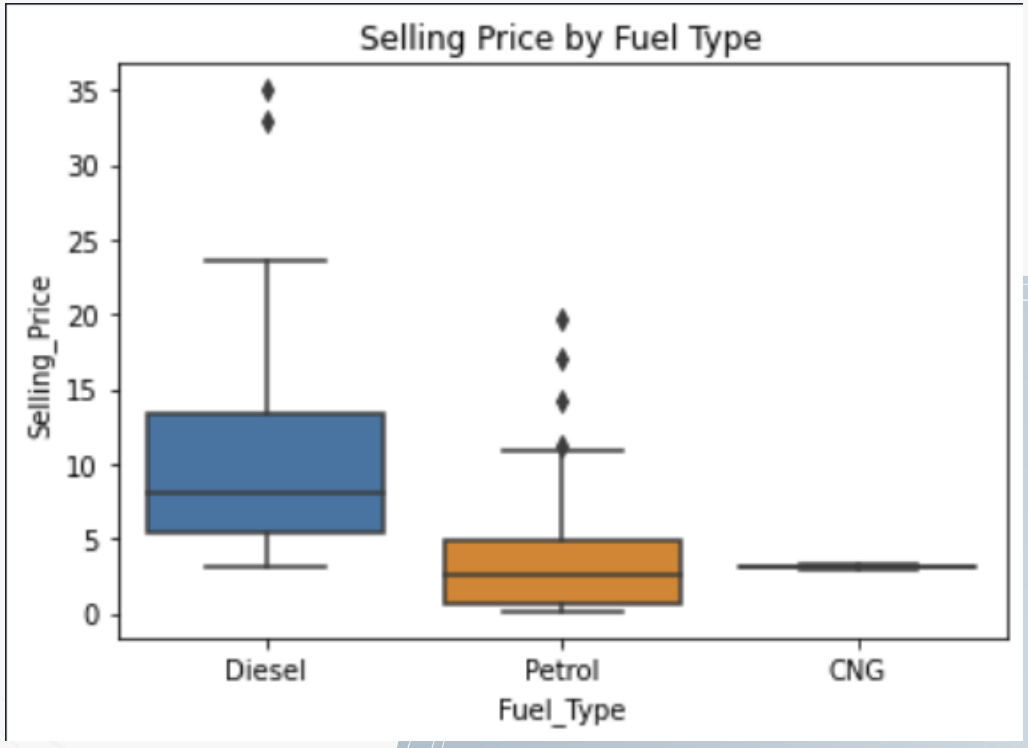
Visualizing Data



Seaborn

- Seaborn is a Python data visualization library that is built on top of Matplotlib.
- It provides a high-level interface for creating informative and attractive statistical graphics. Seaborn includes a range of visualization types, including heatmaps, time series plots, and violin plots.
- It also provides tools for working with categorical data, making it a popular choice for data exploration and analysis.

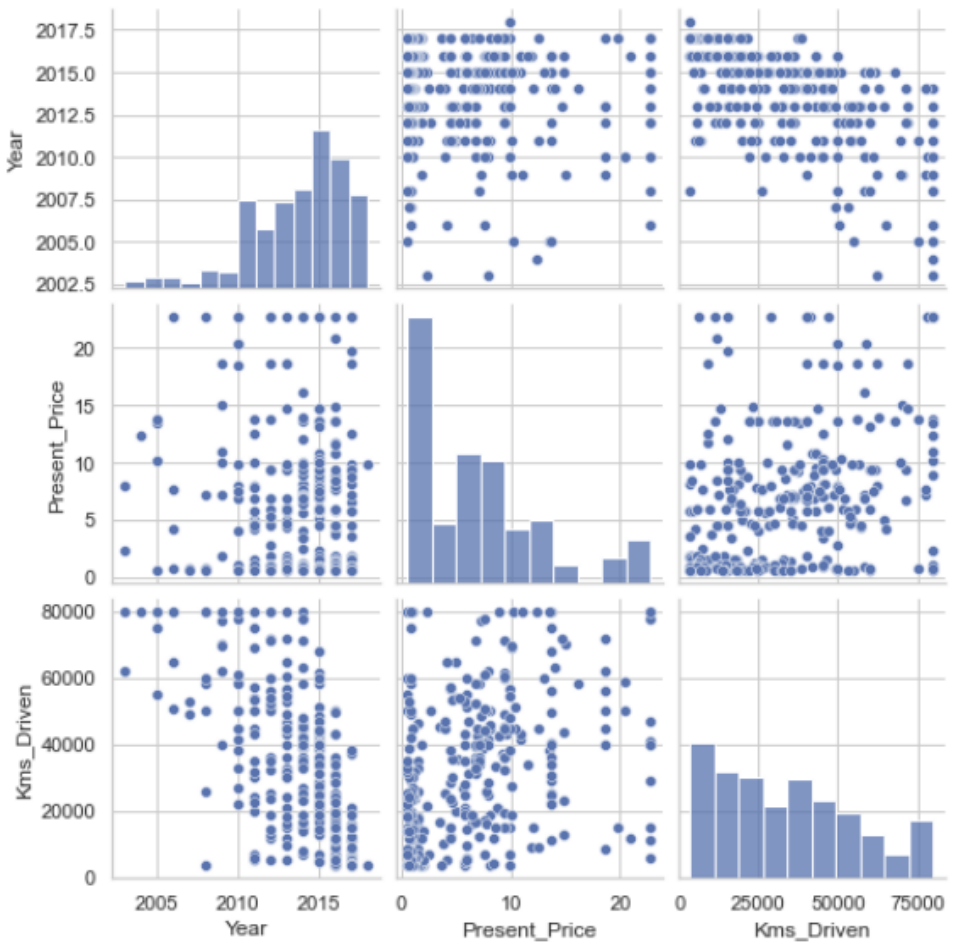
Visualizing Data



Seaborn

- Seaborn is a Python data visualization library that is built on top of Matplotlib.
- It provides a high-level interface for creating informative and attractive statistical graphics. Seaborn includes a range of visualization types, including heatmaps, time series plots, and violin plots.
- It also provides tools for working with categorical data, making it a popular choice for data exploration and analysis.

Visualizing Data



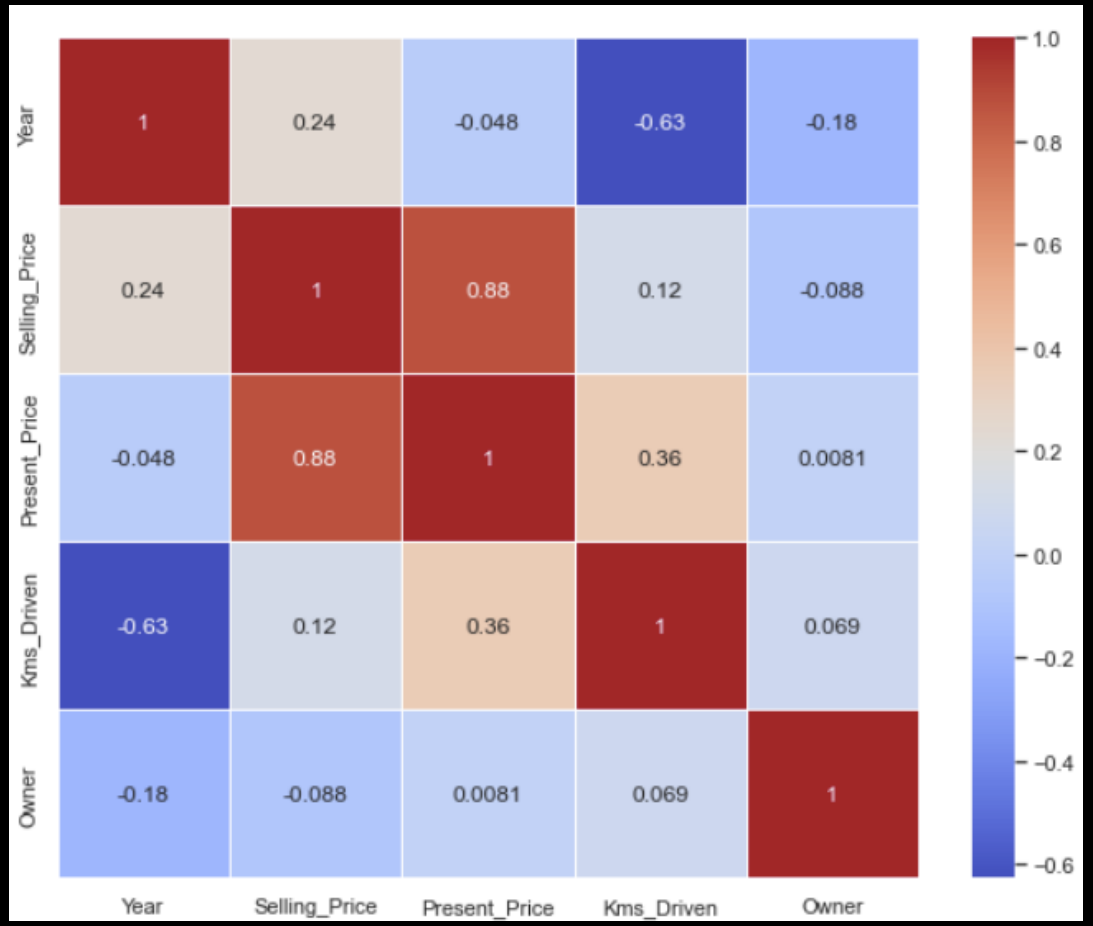
Seaborn – Pair Plot

- Histograms to show distribution of data
- Scatterplots to show relationships

```
# Select numerical columns for the pair plot
numerical_columns = ['Year', 'Present_Price', 'Kms_Driven']

# Create the pair plot
sns.pairplot(data=my_df[numerical_columns])
plt.show()
```

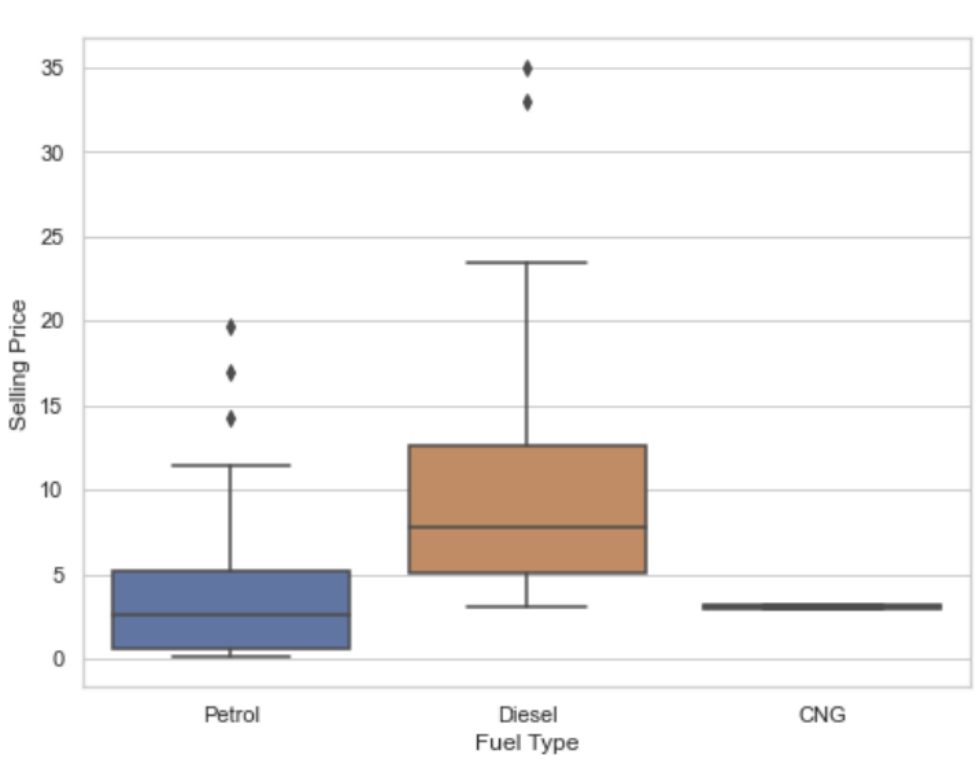
Visualizing Data

- **Seaborn Heatmap**




```python
### Heatmap
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Create a correlation matrix using the DataFrame
correlation_matrix = my_df.corr()

# Create a heatmap using seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap',)
plt.show()
```

Hypothesis Testing



Scipy – t-test

```python
# Hypothesis Testing
from scipy.stats import ttest_ind

fuel_type_petrol = my_df[my_df['Fuel_Type'] == 'Petrol']['Selling_Price']
fuel_type_diesel = my_df[my_df['Fuel_Type'] == 'Diesel']['Selling_Price']

t_stat, p_value = ttest_ind(fuel_type_petrol, fuel_type_diesel)
print("\nHypothesis Testing:")
print("T-statistic:", t_stat)
print("P-value:", p_value)
```

```
In [185]: runcell(0, 'C:/Users/Daniel/.spyder-py3/untitled14.py')

Hypothesis Testing:
T-statistic: -11.407068417938019
P-value: 3.0727821906101835e-25
Reject the null hypothesis. There is a significant difference in Selling Prices based on Fuel Type.
```

Pandas – Save a CSV

```
# Save the DataFrame to a CSV file
my_df.to_csv('output_file.csv', index=False)
```

'Index = False' will drop index column

TIME-SERIES

Time-Series

- What is Time Series Analysis?
  - Time series analysis involves studying data collected over time to identify patterns, trends, and seasonality.
  - Time series analysis helps in understanding past behavior, making predictions, and decision-making.
- Common Use Cases
  - Sales Predictions: Predicting future sales based on historical sales data.
  - Weather Forecasting: Forecasting weather conditions using historical weather data.

Time-Series

- Trend
- Season
- Cycle
- Noise

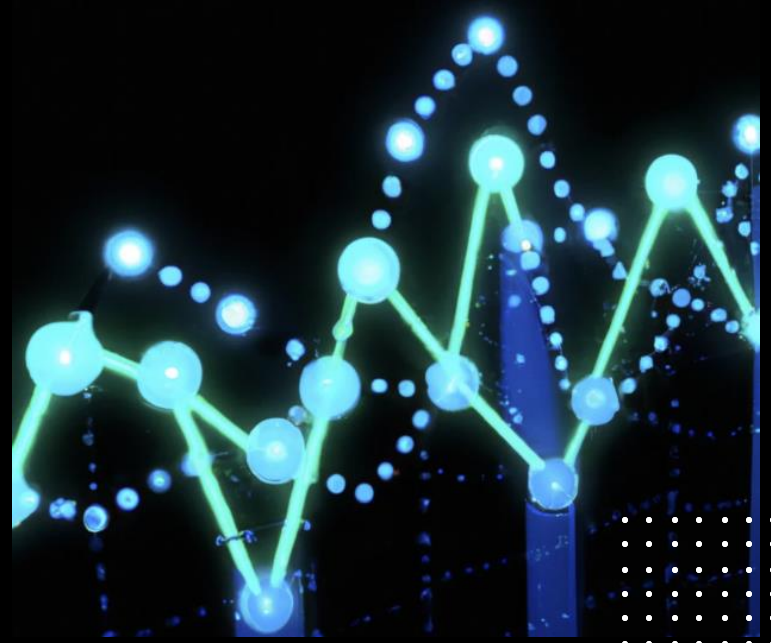
Index	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850	United Kingdom
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26	7.65	17850	United Kingdom
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26	4.25	17850	United Kingdom
7	536366	22633	HAND WARMER UNION JACK	6	12/1/2010 8:28	1.85	17850	United Kingdom

E-Commerce Data

Sales per Day?

E-Commerce Data

- Dataset Overview
  - The dataset consists of customer transactions with details on the items sold.
- Feature Engineering
  - To analyze the daily sales, we need to preprocess the data due to the presence of unit price and quantity sold(missing total).
  - We will derive a new feature by multiplying the unit price and quantity sold to calculate the total sales for each transaction.
- Grouping Data
  - Once the feature engineering is complete, we will group the data by date to aggregate total sales for each day.
  - This will provide us with a time series of daily sales, allowing us to understand sales patterns over time.

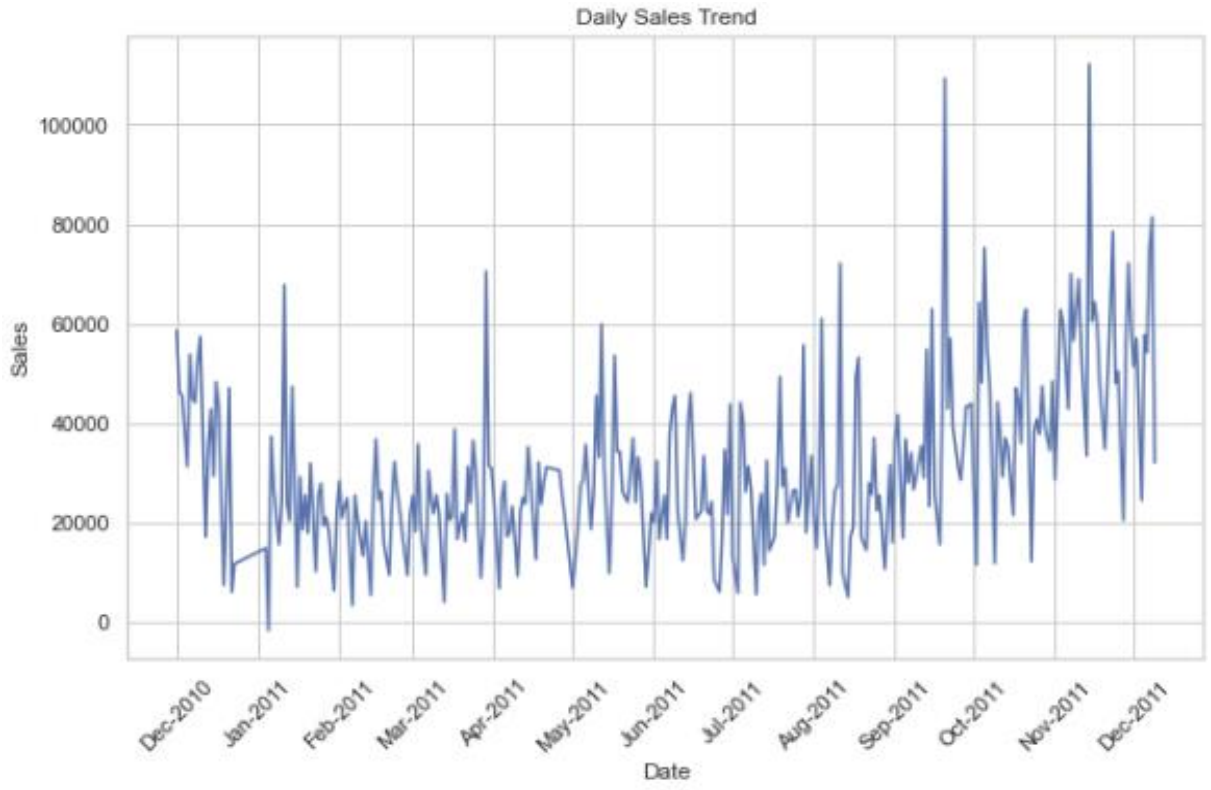


E-Commerce Data

- The dt accessor is used to access the datetime properties of the 'Date' column.

```
# Create new column
df['TotalPrice'] = df.Quantity * df.UnitPrice

# Convert the 'Date' column to datetime data type
df['Date'] = pd.to_datetime(df['InvoiceDate'])

# Group by the 'Date' column and sum the sales for each day
daily_sales = df.groupby(df['Date'].dt.date)['TotalPrice'].sum().reset_index()
```

```python
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.dates import MonthLocator, DateFormatter

# Set the Seaborn style (optional, for aesthetics)
sns.set(style="whitegrid")

# Create the line plot using Seaborn and Matplotlib
plt.figure(figsize=(10, 6))
ax = sns.lineplot(data=daily_sales, x='Date', y='TotalPrice')

plt.title('Daily Sales Trend')
plt.xlabel('Date')
plt.ylabel('Sales')

# Set the x-axis ticker to display only months with 12 evenly spaced ticks
months = MonthLocator(range(1, 13), bymonthday=1, interval=1)
ax.xaxis.set_major_locator(months)

# Format the x-axis labels as 'Month-Year'
date_format = DateFormatter("%b-%Y")
ax.xaxis.set_major_formatter(date_format)

plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.show()
```



E-Commerce Data

```python
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.dates import MonthLocator, DateFormatter

# Set the Seaborn style (optional, for aesthetics)
sns.set(style="whitegrid")

# Create the line plot using Seaborn and Matplotlib
plt.figure(figsize=(10, 6))
ax = sns.lineplot(data=daily_sales, x='Date', y='TotalPrice')

plt.title('Daily Sales Trend')
plt.xlabel('Date')
plt.ylabel('Sales')

# Set the x-axis ticker to display only months with 12 evenly spaced ticks
months = MonthLocator(range(1, 13), bymonthday=1, interval=1)
ax.xaxis.set_major_locator(months)

# Format the x-axis labels as 'Month-Year'
date_format = DateFormatter("%b-%Y")
ax.xaxis.set_major_formatter(date_format)

plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.show()
```

- MonthLocator: Class from the matplotlib.dates module that helps determine the locations of major tick marks on the x-axis corresponding to months.
- bymonthday=1: This specifies that we want the major tick marks to be placed on the 1st day of each month.
- interval=1: This indicates that we want the ticks to be spaced every 1 month.

- %b: Represents the abbreviated month name (e.g., Jan, Feb, Mar).
- -: separator.
- %Y: Represents the full year with four digits (e.g., 2023).

E-Commerce Data

MACHINE LEARNING

Machine Learning

Types

- Linear regression
- Logistic regression
- Decision trees
- Random forests
- Neural networks

Machine Learning

Linear Regression

- A model that predicts a continuous output based on one or more input variables. It assumes a linear relationship between the input and output variables.

Machine Learning

Logistic Regression

- Logistic regression: A model that predicts a binary output (e.g., yes or no) based on one or more input variables. It uses a sigmoid function to convert the output of a linear regression into a probability value.

Machine Learning

Decision trees

- Decision trees: A model that uses a tree-like structure to make predictions based on a series of if-then statements. Each node in the tree represents a decision based on a feature, and each leaf node represents a final decision or outcome.

Machine Learning

Random forests

- A collection of decision trees that work together to make predictions. Each tree is trained on a random subset of the data and features, and the final prediction is based on the average of the predictions from all the trees.

Machine Learning

Neural networks

- Neural networks: A model that uses layers of interconnected nodes to learn complex relationships between input and output data. It can be used for both regression and classification tasks.

Machine Learning – Neural Networks

- Neural networks: A model that uses layers of interconnected nodes to learn complex relationships between input and output data. It can be used for both regression and classification tasks.

PICKING A MODEL

SKLEARN

Sklearn

- Scikit-learn (or sklearn for short) is a popular open-source machine learning library for Python.
- It provides a wide range of algorithms and tools for various machine learning tasks such as classification, regression, clustering, and dimensionality reduction. Sklearn is built on top of NumPy, SciPy, and Matplotlib, and it integrates well with other Python libraries.
- Sklearn is easy to use and well documented, making it a popular choice for beginners and experts alike.

Sklearn – Random Forest Regression Example

```
### Use sklearn to make a random forest model to predict current price

# Import necessary libraries

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Convert categorical variables into dummy variables
my_df = pd.get_dummies(my_df, columns=['Car_Name','Fuel_Type', 'Seller_Type', 'Transmission'], drop_first=True)

# Prepare the my_df for the model
X = my_df.drop(['Present_Price'], axis=1) # Feature matrix
y = my_df['Present_Price'] # Target variable

# Split the my_df into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the random forest model
rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training my_df
rf.fit(X_train, y_train)

# Predict on the test set
y_pred = rf.predict(X_test)
```

- Scikit-learn (or sklearn for short) is a popular open-source machine learning library for Python.
- It provides a wide range of algorithms and tools for various machine learning tasks such as classification, regression, clustering, and dimensionality reduction. Sklearn is built on top of NumPy, SciPy, and Matplotlib, and it integrates well with other Python libraries.
- Sklearn is easy to use and well documented, making it a popular choice for beginners and experts alike.

Sklearn – dummy variables

```
### Use sklearn to make a random forest model to predict current price

# Import necessary libraries

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Convert categorical variables into dummy variables
my_df = pd.get_dummies(my_df, columns=['Car_Name','Fuel_Type', 'Seller_Type', 'Transmission'], drop_first=True)

# Prepare the my_df for the model
X = my_df.drop(['Present_Price'], axis=1) # Feature matrix
y = my_df['Present_Price'] # Target variable

# Split the my_df into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the random forest model
rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training my_df
rf.fit(X_train, y_train)

# Predict on the test set
y_pred = rf.predict(X_test)
```

- The `drop_first=True` argument drops the first column of each set of dummy variables to avoid multicollinearity.
- Multicollinearity is a situation where two or more predictor variables in a regression model are highly correlated with each other.
- Multicollinearity can cause issues with the stability and interpretability of the model, and it can also make it difficult to determine the individual effects of each variable on the response variable.

Sklearn – Train/Test

```
### Use sklearn to make a random forest model to predict current price

# Import necessary libraries

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Convert categorical variables into dummy variables
my_df = pd.get_dummies(my_df, columns=['Car_Name','Fuel_Type', 'Seller_Type', 'Transmission'], drop_first=True)

# Prepare the my_df for the model
X = my_df.drop(['Present_Price'], axis=1) # Feature matrix
y = my_df['Present_Price'] # Target variable

# Split the my_df into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the random forest model
rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training my_df
rf.fit(X_train, y_train)

# Predict on the test set
y_pred = rf.predict(X_test)
```

- In machine learning, we typically split our data into two subsets: a training set and a testing set.
- The training set is used to train our machine learning model, while the testing set is used to evaluate the performance of the trained model on new, unseen data.
- This is an important step in developing a machine learning model that can accurately generalize to new data.

Sklearn – Train/Test

```
### Use sklearn to make a random forest model to predict current price

# Import necessary libraries

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Convert categorical variables into dummy variables
my_df = pd.get_dummies(my_df, columns=['Car_Name','Fuel_Type', 'Seller_Type', 'Transmission'], drop_first=True)

# Prepare the my_df for the model
X = my_df.drop(['Present_Price'], axis=1) # Feature matrix
y = my_df['Present_Price'] # Target variable

# Split the my_df into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the random forest model
rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training my_df
rf.fit(X_train, y_train)

# Predict on the test set
y_pred = rf.predict(X_test)
```

Considerations

- Data Quality: It's essential to ensure that the data used for training and testing is of high quality and representative of the population we want to make predictions on. Poor quality data can lead to inaccurate model predictions.
- Randomness: It's important to ensure that the data is split into training and testing sets in a random manner to avoid any bias in the model's performance.
- Size of the Data: The size of the training and testing data is another important consideration.

Sklearn – Use

```
# Test model on a random value

# Randomly select a row from the original DataFrame
test_row = my_df.sample(1)

# Drop the Present_Price column from the selected row
new_input = test_row.drop('Present_Price', axis=1)

# Use the trained model to make a prediction on the new input
predicted_price = rf.predict(new_input)

# Print the original and predicted selling prices
print('Original Present Price:', test_row['Present_Price'].values[0])
print('Predicted Present Price:', predicted_price[0])
```

```
Original Present Price: 9.4
Predicted Present Price: 9.234599999999986
```

- Scikit-learn (or sklearn for short) is a popular open-source machine learning library for Python.
- It provides a wide range of algorithms and tools for various machine learning tasks such as classification, regression, clustering, and dimensionality reduction. Sklearn is built on top of NumPy, SciPy, and Matplotlib, and it integrates well with other Python libraries.
- Sklearn is easy to use and well documented, making it a popular choice for beginners and experts alike.

Sklearn – Hyperparamaters

```
# Test model on a random value

# Randomly select a row from the original DataFrame
test_row = my_df.sample(1)

# Drop the Present_Price column from the selected row
new_input = test_row.drop('Present_Price', axis=1)

# Use the trained model to make a prediction on the new input
predicted_price = rf.predict(new_input)

# Print the original and predicted selling prices
print('Original Present Price:', test_row['Present_Price'].values[0])
print('Predicted Present Price:', predicted_price[0])
```

- Hyperparameters are parameters that are set prior to training the model, such as learning rate, batch size, number of hidden layers, and regularization strength.

Sklearn – Hypertuning

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Create a Random Forest Regressor object
rf = RandomForestRegressor()

# Define the hyperparameters to tune
params = {'n_estimators': [50, 100, 150, 200],
          'max_depth': [5, 10, 15, 20],
          'min_samples_split': [2, 5, 10],
          'min_samples_leaf': [1, 2, 4],
          'max_features': ['sqrt', 'log2']}

# Perform grid search cross-validation to find the optimal hyperparameters
grid_search = GridSearchCV(estimator=rf, param_grid=params, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters found
print("Best hyperparameters: ", grid_search.best_params_)

# Predict the test data using the best model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

- Cross-validation is a technique used to evaluate machine learning models by partitioning the data into training and testing sets multiple times. This allows us to assess the model's performance on different subsets of the data and can help prevent overfitting, which occurs when a model fits too closely to the training data and fails to generalize to new data.

Sklearn – Hypertuning

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Create a Random Forest Regressor object
rf = RandomForestRegressor()

# Define the hyperparameters to tune
params = {'n_estimators': [50, 100, 150, 200],
          'max_depth': [5, 10, 15, 20],
          'min_samples_split': [2, 5, 10],
          'min_samples_leaf': [1, 2, 4],
          'max_features': ['sqrt', 'log2']}

# Perform grid search cross-validation to find the optimal hyperparameters
grid_search = GridSearchCV(estimator=rf, param_grid=params, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters found
print("Best hyperparameters: ", grid_search.best_params_)

# Predict the test data using the best model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

```
Best hyperparameters:  {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 150}
```

Sklearn – Random Forest Binary Classification Example

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


# Encode the categorical variables
le = LabelEncoder()
my_df['Fuel_Type'] = le.fit_transform(my_df['Fuel_Type'])
my_df['Seller_Type'] = le.fit_transform(my_df['Seller_Type'])
my_df['Transmission'] = le.fit_transform(my_df['Transmission'])

# Split the dataset into training and testing sets
X = my_df.drop(['Car_Name', 'Transmission'], axis=1)
y = my_df['Transmission']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Random Forest Classifier object
rf = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)

# Fit the model to the training data
rf.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = rf.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

- Prediting binary outcome, categorical data.
- Regression used for continuous data.

EVALUATION METRICS

Classification Metrics

```
from sklearn.metrics import classification_report

predictions = rf.predict(X_test)
report = classification_report(y_test, predictions)
print("Classification Report:\n", report)
```

```
In [27]: runcell(28, 'C:/Users/Daniel/.spyder-py3/python_data_science_class_notes.py')
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.36      0.50        11
           1       0.88      0.98      0.92        50

    accuracy                           0.87        61
   macro avg       0.84      0.67      0.71        61
weighted avg       0.86      0.87      0.85        61
```

- Classification Report
  - Accuracy
  - Support - Total instances of each category

Classification Metrics

```
from sklearn.metrics import roc_auc_score

predictions = rf.predict(X_test)
roc = roc_auc_score(y_test, predictions)
print("AUC Score:\n",roc)
```

```
In [29]: runcell(29, 'C:/Users/Daniel/.spyder-py3/python_data_science_class_notes.py')
AUC Score:
 0.6718181818181819
```

- Roc_auc
  - 1 = model is perfect
  - .5 = model is worthless

Classification Metrics

```
from sklearn.metrics import confusion_matrix

predictions = rf.predict(X_test)
cm = confusion_matrix(y_test, predictions)
print("Confusion Matrix:\n",cm)
```

```
In [26]: runcell(27, 'C:/Users/Daniel/.spyder-py3/python_data_science_class_notes.py')
Confusion Matrix:
 [[ 4  7]
 [ 1 49]]
```

- Confusion Matrix

- `[true_negative, false_positive]`
- `[false_negative, true_positive]`

Regression Metrics

- Mean Squared Error

```
from sklearn.metrics import mean_squared_error

# Assume reg is your regressor
predictions = rf.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
In [32]: runcell(30, 'C:/Users/Daniel/.spyder-py3/python_data_science_class_notes.py')
Mean Squared Error: 5.785082772040986
```

Regression Metrics

- R squared

```
from sklearn.metrics import r2_score

predictions = rf.predict(X_test)
r2 = r2_score(y_test, predictions)
print("R² Score:", r2)
```

```
In [33]: runcell(31, 'C:/Users/Daniel/.spyder-py3/python_data_science_class_notes.py')
R² Score: 0.8853000598807674
```

Extra Resources

- https://Kaggle.com
- https://cs229.stanford.edu/

THANKYOU

Daniel Bissell

Danielabissell@gmail.com

- https://www.linkedin.com/in/daniel-bissell-3b0b79bb/