Data Science with Python Career Program - Capstone Project

- By Sumit S. Chaure
  (Batch – 10)

Agenda

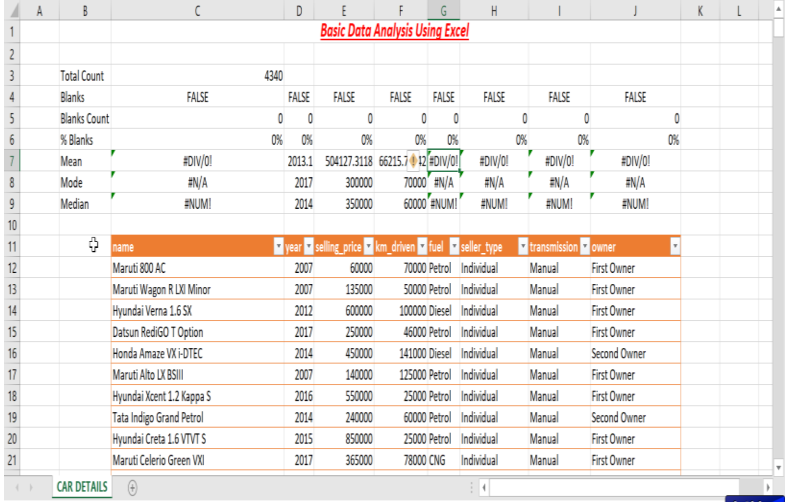
skill academy

- Data Exploration (Using Excel & Python)
- Data insights (Excel & Python)
- EDA Graphs
- Graphical Analysis and conclusion on Data
- Data Cleaning & Pre-Processing Steps
- ML Modeling
- Model Test Evaulation & Prediction Analysis
- Deployment of ML Models using Streamlit

Data Exploration (Using Excel)

skill academy

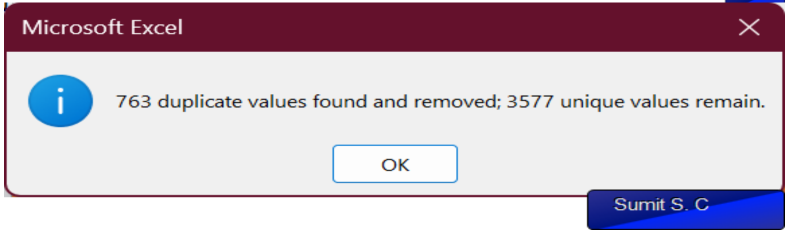
Basic Analysis Using Excel Sheet



Basic Data Analysis Using Excel

	C	D	E	F	G	H	I	J
Total Count		4340						
Blanks	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Blanks Count	0	0	0	0	0	0	0	0
% Blanks	0%	0%	0%	0%	0%	0%	0%	0%
Mean	#DIV/0!	2013.1	504127.3118	66215.7[illegible]12	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
Mode	#N/A	2017	300000	70000	#N/A	#N/A	#N/A	#N/A
Median	#NUM!	2014	350000	60000	#NUM!	#NUM!	#NUM!	#NUM!

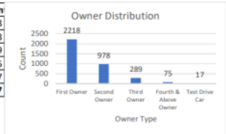
name	year	selling_price	km_driven	fuel	seller_type	transmission	owner
Maruti 800 AC	2007	60000	70000	Petrol	Individual	Manual	First Owner
Maruti Wagon R LXI Minor	2007	135000	50000	Petrol	Individual	Manual	First Owner
Hyundai Verna 1.6 SX	2012	600000	100000	Diesel	Individual	Manual	First Owner
Datsun RediGO T Option	2017	250000	46000	Petrol	Individual	Manual	First Owner
Honda Amaze VX i-DTEC	2014	450000	141000	Diesel	Individual	Manual	Second Owner
Maruti Alto LX BSIII	2007	140000	125000	Petrol	Individual	Manual	First Owner
Hyundai Xcent 1.2 Kappa S	2016	550000	25000	Petrol	Individual	Manual	First Owner
Tata Indigo Grand Petrol	2014	240000	60000	Petrol	Individual	Manual	Second Owner
Hyundai Creta 1.6 VTVT S	2015	850000	25000	Petrol	Individual	Manual	First Owner
Maruti Celerio Green VXI	2017	365000	78000	CNG	Individual	Manual	First Owner



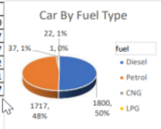
- The Dataset contains **8 rows & 4240** columns showing the information of used cars dataset.
- The Basic analysis shows that there *is no null value* present.
- The Analysis further shows that there are *763 duplicate* values.

Basic Pivot Analysis

Owners	Owner Count
First Owner	2218
Second Owner	978
Third Owner	289
Fourth & Above	75
Test Drive Car	17
Grand Total	3577

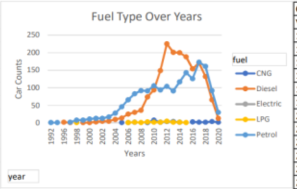


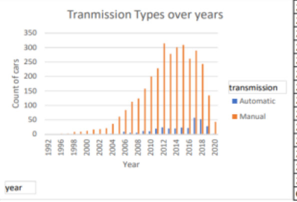
Fuel Type	Count
Diesel	1800
Petrol	1717
CNG	37
LPG	22
Electric	1
Grand Total	3577



pe	Count
k Deale	33
l	712
	2832
otal	3577

Count of name	Fuel type					
Year	CNG	Diesel	Electric	LPG	Petrol	Grand Total
1992					1	1
1995					1	1
1996		2				2
1997		1			2	3
1998				1	8	9
1999		1			8	9
2000		1			11	12
2001		3			13	16
2002		5			13	18
2003		5			17	22
2004		10			28	38
2005	1	14			46	61
2006		25	1	1	66	93
2007	2	30		2	83	117
2008		36		1	92	129
2009	1	74		3	91	169
2010	8	94		2	106	210
2011	2	149		2	94	247
2012	4	225		5	104	338
2013	4	201		2	91	298
2014	2	200		2	117	321
2015		188		1	143	332
2016	3	154			126	283
2017	2	171			173	346
2018	2	132			161	295
2019	4	66			92	162
2020	2	13			30	45
Grand Total	37	1800	1	22	1717	3577





Count of name	Tranmission type		
Years	Automatic	Manual	Grand Total
1992		1	1
1995		1	1
1996		2	2
1997	1	2	3
1998	1	8	9
1999		9	9
2000		12	12
2001		16	16
2002		18	18
2003	1	21	22
2004	2	36	38
2005		61	61
2006	9	84	93
2007	5	112	117
2008	5	124	129
2009	11	158	169
2010	10	200	210
2011	19	228	247
2012	24	314	338
2013	20	278	298
2014	20	301	321
2015	23	309	332
2016	22	261	283
2017	57	289	346
2018	52	243	295
2019	28	134	162
2020	2	43	45
Grand Total	312	3265	3577

Sumit S. C

Data Exploration (Using Python)

skill academy

Step 1 – Import Library & Read File

2. Module Imports

Click here to ask Blackbox to help you code faster

```python
import pandas as pd              # for data cleaning and data pre-processing, CSV file I/O,etc
import numpy as np               # linear algebra & for mathematical computation
import matplotlib.pyplot as plt  # for visualization
%matplotlib inline
import seaborn as sns            # for visualization
from collections import Counter  # to count occurrences
from tabulate import tabulate    # to make tables for results

import warnings                  # for warning removals in code output
warnings.filterwarnings('ignore')
```

2.1) Importing the dataset (With error handling)

Click here to ask Blackbox to help you code faster

```python
# 2.1) Importing the dataset (With error handling)
# If you want to upload the dataset directly (Since on Google Colab it will be lost on re-run) - uncomment the below 2 line code and run
# from google.colab import files
# uploaded = files.upload()

file_path = "../data/raw/CAR DETAILS.csv"
file_name = file_path.split("/")[-1]

try:
    # Reading the CSV file into a Pandas DataFrame
    df = pd.read_csv(file_path)
    # Store the filename as an attribute in the DataFrame
    df.file_name = file_name
    print(f"\n '{df.file_name}' loaded successfully.")

# Exception to check if the file has some error like no file at the path, etc.
except FileNotFoundError:
    print(f"Error: '{file_name}' not found at the specified location {
          file_path}.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

Python

```
'CAR DETAILS.csv' loaded successfully.
```

Step 2 – Basic Data Lookup

Click here to ask Blackbox to help you code faster

```python
df
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner
0	Maruti 800 AC	2007	60000	70000	Petrol	Individual	Manual	First Owner
1	Maruti Wagon R LXI Minor	2007	135000	50000	Petrol	Individual	Manual	First Owner
2	Hyundai Verna 1.6 SX	2012	600000	100000	Diesel	Individual	Manual	First Owner

df – loads the data frame we gave while loading the csv & displays data.

2.2) Showing Basic Dataset Information

Click here to ask Blackbox to help you code faster

```python
# Check the shape of the DataFrame
print("\nShape of the DataFrame:")
print(df.shape)
print(df.size)
num_rows, num_columns = df.shape
print(f"Rows: {num_rows}, Columns: {num_columns}")

# Display information about the dataset
print(f"\nDataset information for {df.file_name}:")
df.head(3)
df.tail(3)
print("\nDataset information:")
print(df.info())
```

```
Shape of the DataFrame:
(4340, 8)
34720
Rows: 4340, Columns: 8

Dataset information for CAR DETAILS.csv:

Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           4340 non-null   object
 1   year           4340 non-null   int64
 2   selling_price  4340 non-null   int64
 3   km_driven      4340 non-null   int64
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
```

df.shape & df.size – gives the rows & column info

df.head & df.tail – displays 5 values from top & bottom of dataset

df.info – gives the information about the column its datatype and null values.

Data Insights (Using Python)

Step 3 – Data Checks

```python
# Null values
print("Missing values in the dataset:\n")
print(df.isnull().sum())

# NA value calculation
nullval = df.isna().sum()
nullval = nullval[nullval > 0]
print("\nSum of Missing values:\n", nullval)

# Calculating the number of missing values or null values in df
total_missing_values = df_dup_dropped.isnull().sum().sum()
print("The number of missing values/NA in dataframe :", total_missing_values)

# Calculate the total number of values in df (excluding the missing values)
total_values = df_dup_dropped.size
print("Total number of values in dataframe :", total_values)
# percentage of missing values or null values in df
percentage_missing_values = (total_missing_values / total_values) * 100
print("Percentage of missing values or null values in df :",
      percentage_missing_values)

# Duplicate values
print("\nChecking for duplicated values:\n")
print(df.duplicated())
print("\nSum of Duplicated Values in Dataframe :", df.duplicated().sum())
# Dropping Duplicate values from dataframe
df_dup_dropped = df.drop_duplicates().reset_index(drop=True)
print("\nSum of Duplicated Values in Dataframe :",
      df_dup_dropped.duplicated().sum())
print(f"Original Dataframe Shape : {df.shape}\nNew Dataframe Shape after Dropping Duplicates: {
      df_dup_dropped.shape}
      \nNo of duplicate rows removed :{df.duplicated().sum()}")

# Value check
print("Value counts of the dataset by datatypes")
df_dup_dropped.dtypes.value_counts()
# Unique values
print("Unique Value counts inside each columns")
df_dup_dropped.nunique()

# Datatype check
print("Data types of each column:")
df_dup_dropped.dtypes

# Category Column Check
df + df_dup_dropped.select_dtypes(include="category")
categorical_columns = df_dup_dropped.select_dtypes(include=["object"]).columns
print("\nCategorical columns:")
print(categorical_columns)

# Summary STats
print("\nSummary statistics:\n")
print(df_dup_dropped.describe())

# Descriptive stats
print(f"The descriptive Stats for the {file_name} dataset:")
df_dup_dropped.describe()
```

Step 4 – Data Pre-Processing

Extracting Car Manufacturer, Model & Variant Name from Car Name (To do more In-depth analysis)

Click here to ask Blackbox to help you code faster

```python
# Unique values in name column
print(f"Unique Values in 'name' column : {df_dup_dropped["name"].nunique()}")    "nunique": Unknown word.
```

✓ 0.0s

```
Unique Values in 'name' column : 1491
```

Click here to ask Blackbox to help you code faster

```python
# Creating a copy to perform the splitting
df_name_split = df_dup_dropped.copy()
# name of the column to be split
car_names = df_dup_dropped["name"]
# Functions to split car names into brand,model & variant
def split_car_name(name):
    words = name.split()
    if len(words) == 3:
        brand, model, variant = words
    else:
        brand = words[0]
        model = ""
        variant = ""
        for i, word in enumerate(words[1:]):
            if word[0].isdigit():
                if i == 0:  # Number at the second position
                    model = " ".join(words[1:2])
                    variant = " ".join(words[2:])
                else:  # Number at the third position or later
                    model = " ".join(words[1 : i + 1])
                    variant = " ".join(words[i + 1 :])
                break
        else:
            model = " ".join(words[1:-1])
            variant = words[-1]
    return brand, model, variant
# Apply the split_car_name function to create new columns
df_name_split[["brand", "model", "variant"]] = pd.DataFrame(car_names.apply(split_car_name).tolist())
# Drop the Name column
df_name_split.drop(columns=["name"], inplace=True)
```

✓ 0.0s

Click here to ask Blackbox to help you code faster

```python
df_name_split.head()
```

✓ 0.0s Open 'df_name_split'

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	brand	model	variant
0	2007	60000	70000	Petrol	Individual	Manual	First Owner	Maruti	800	AC

- ***Step 3*** – We do basic checks to find *null, duplicates, NA, unique values* in dataset.
- We also find the categorical columns & statistics to get info about numerical data.
- **Step 4** – We *Pre-process the data* and handle the features like splitting name into model, brand, variant to do analysis.
- Dropped the name column after split.

Data Insights (Using Python)

skill academy

Step 5 – Adjusting Data

Adjusting Column Names

Click here to ask Blackbox to help you code faster

```
df_name_split.head(1)
```

0.0s Open 'df_name_split'

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	brand	model	variant
0	2007	60000	70000	Petrol	Individual	Manual	First Owner	Maruti	800	AC

Click here to ask Blackbox to help you code faster

```
brand_name = df_name_split["brand"]
model_name = df_name_split["model"]
variant_name = df_name_split["variant"]
selling_price = df_name_split["selling_price"]
df_name_split.drop(["brand", "model", "variant","selling_price"], axis=1, inplace=True)
df_name_split.insert(0, "brand", brand_name)
df_name_split.insert(1, "model", model_name)
df_name_split.insert(2, "variant", variant_name)
df_name_split.insert(len(df_name_split.columns),"selling_price", selling_price)
```

0.0s

Click here to ask Blackbox to help you code faster

```
df_name_split.head(1)
```

0.0s Open 'df_name_split'

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	brand	model	variant
0	2007	60000	70000	Petrol	Individual	Manual	First Owner	Maruti	800	AC

Click here to ask Blackbox to help you code faster

```
df_name_split.tail(1)
```

0.0s Open 'df_name_split'

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	brand	model	variant
3576	2016	225000	40000	Petrol	Individual	Manual	First Owner	Renault	KWID	RXT

Changing the case of column names

Click here to ask Blackbox to help you code faster

```
df_changed = df_name_split.copy()
df_changed.columns = df_changed.columns.str.title()
df_changed.head(1)
```

0.0s Open 'df_changed'

Year	Selling_Price	Km_Driven	Fuel	Seller_Type	Transmission	Owner	Brand	Model	Variant

Insights :

- Data Exploration is very useful step in the process of analytics and model building as it helps us to remove any redundant data & also construct new features to enhance the analysis and model building.
- In the data exploration part we eye-ball the data to look at features that help us in analysis and model formation and help us to remove any unnecessary junk from our data.
- Most times the data from sources is not cleaned so data cleaning is an important part of the analysis work.
- Gathering and exploring data and finding meaningful insights helps us to analyse and train the model more efficiently.
- In the step given on the side we adjusted the column names such that the newly added feature at the end are given index value & adjusted likewise selling price being target we move it to end.

Data insights

skill academy

Basic Info On Dataset

```
Shape of the DataFrame:
(4340, 8)
34720
Rows: 4340, Columns: 8

Dataset information for CAR DETAILS.csv:

Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           4340 non-null   object
 1   year           4340 non-null   int64
 2   selling_price  4340 non-null   int64
 3   km_driven      4340 non-null   int64
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
None
```

The dataset contains ***4340 rows & 8 columns*** , all are non-null values *3 are numerical* while *5 are categorical type.*

Dataset after cleaning & adding features

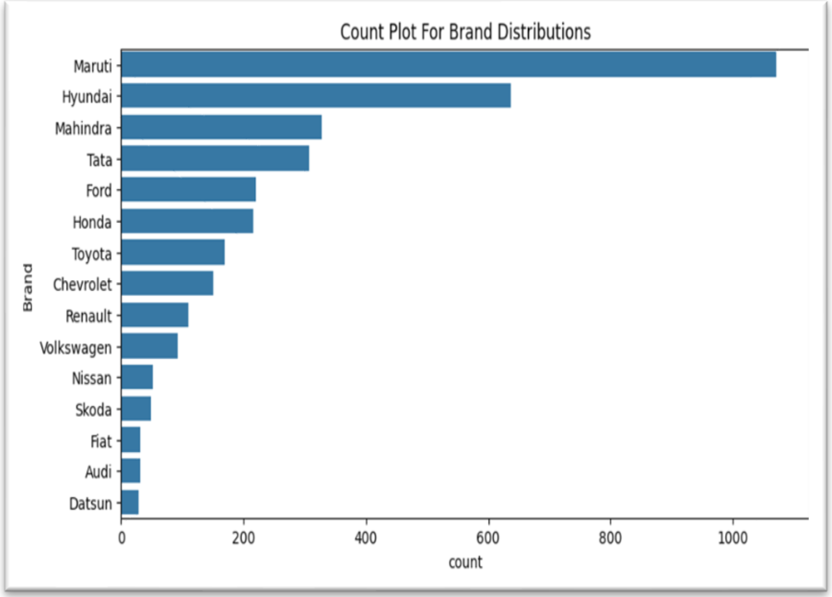
```
Shape of the DataFrame:
(3577, 10)
35770
Rows: 3577, Columns: 10

Dataset information for CAR DETAILS.csv:

Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3577 entries, 0 to 3576
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Year           3577 non-null   int64
 1   Selling_Price  3577 non-null   int64
 2   Km_Driven      3577 non-null   int64
 3   Fuel           3577 non-null   object
 4   Seller_Type    3577 non-null   object
 5   Transmission   3577 non-null   object
 6   Owner          3577 non-null   object
 7   Brand          3577 non-null   object
 8   Model          3577 non-null   object
 9   Variant        3577 non-null   object
dtypes: int64(3), object(7)
memory usage: 279.6+ KB
None
```

After cleaning the data , removing duplicates and adding removing features from dataset the new processed dataset contains ***3577 rows & 10 columns***.*3 are numerical* while *7 are categorical* columns.

EDA Graphs.

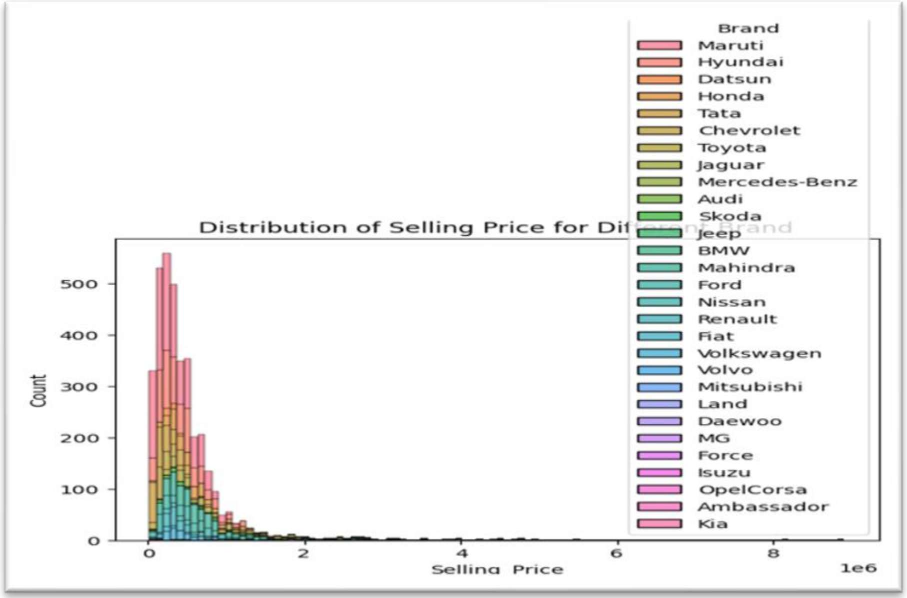
The graphs below are made using matplotlib & Seaborn library (code in notebook file)

Count Plot



Count-plot showing the distribution of cars according to brand

Hist Plot

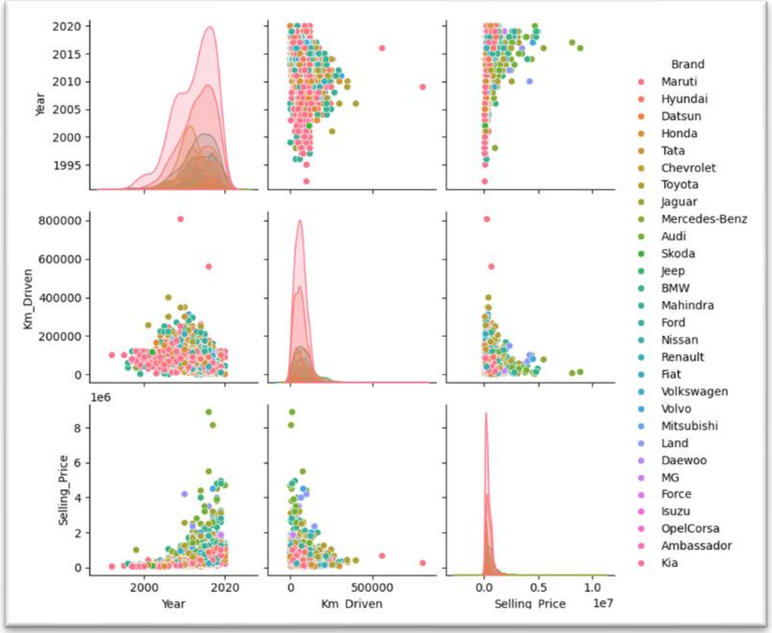


Hist-plot to showcase the selling price of various brands

EDA Graphs.

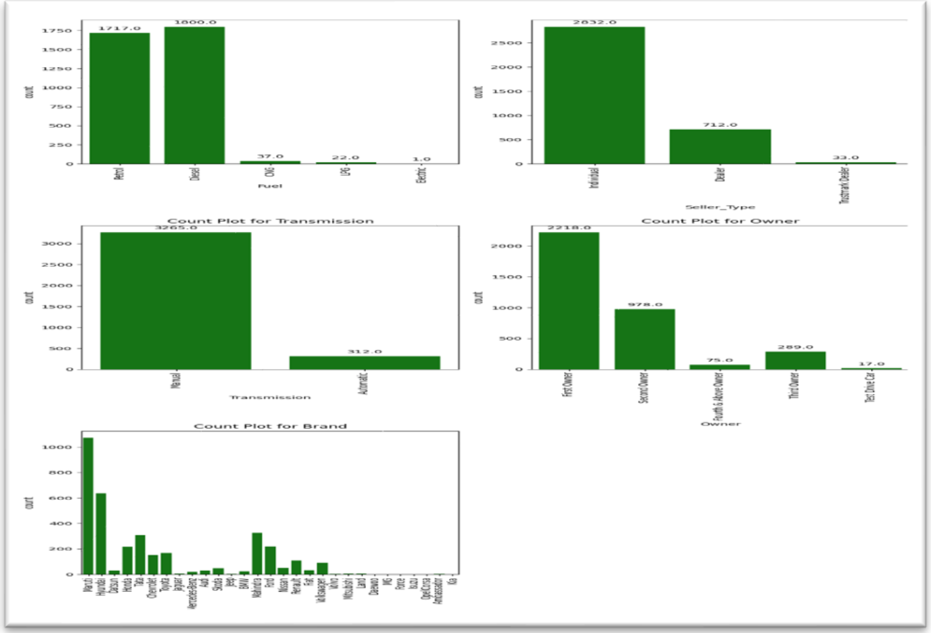
skill academy

Pair Plot (Numerical Data)



Pair-plot showing the distribution of numerical data(selling price, km_driven, years) according to brand

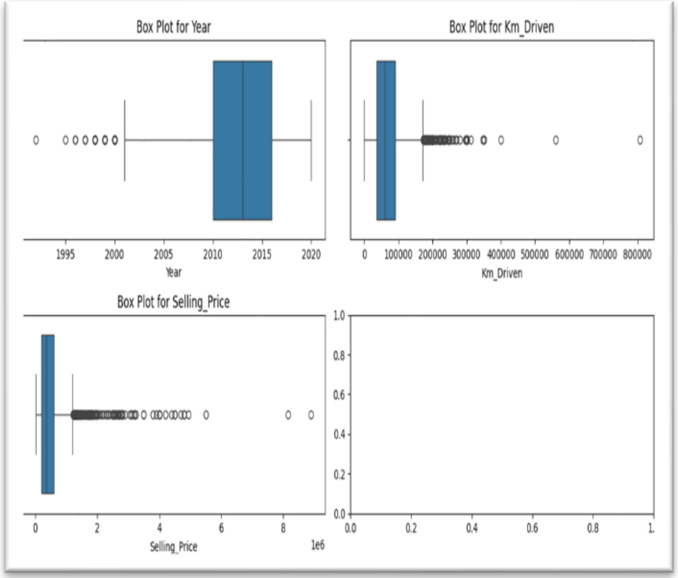
Count Plot (Categorical Data

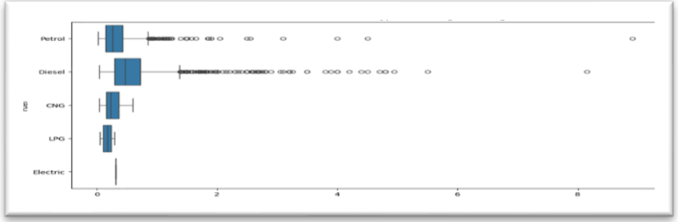


Count Plot of various categorical data representation.

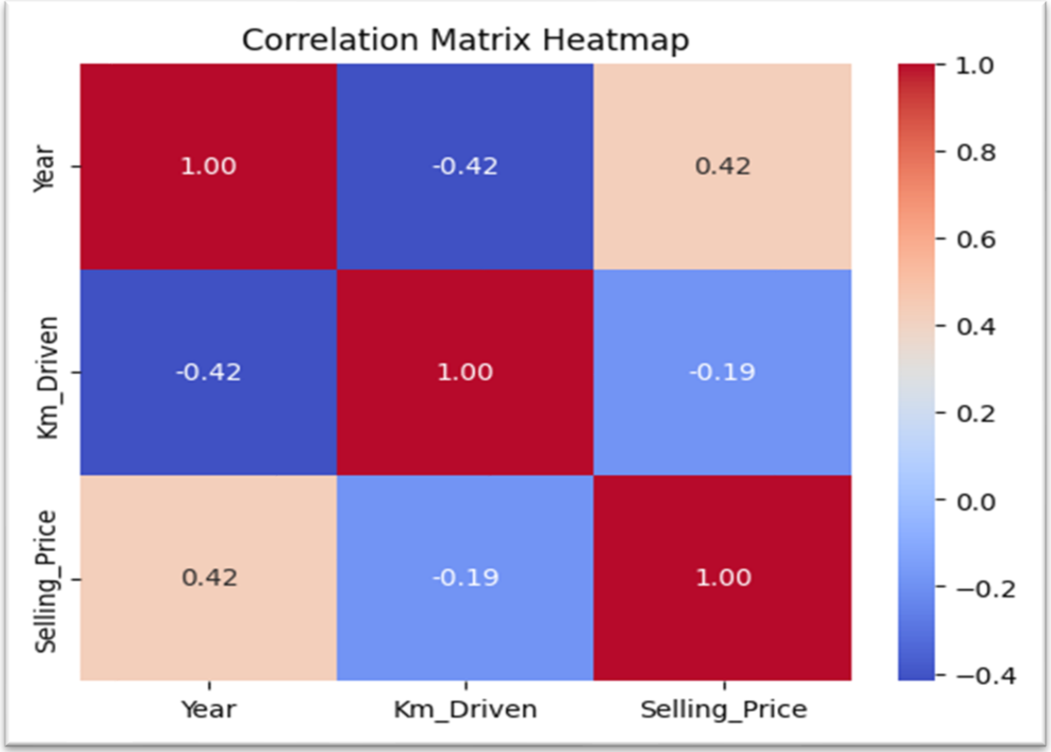
Graphical Analysis and conclusion on Data

Box Plot to find the Outlier data



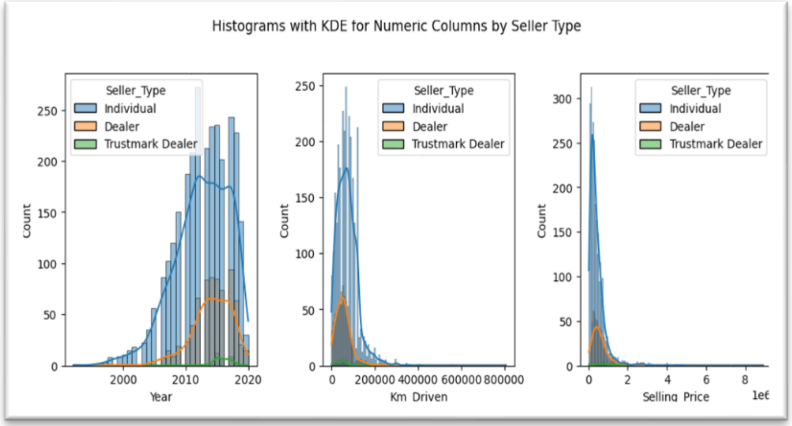


HeatMap to find the Correlation in Numerical

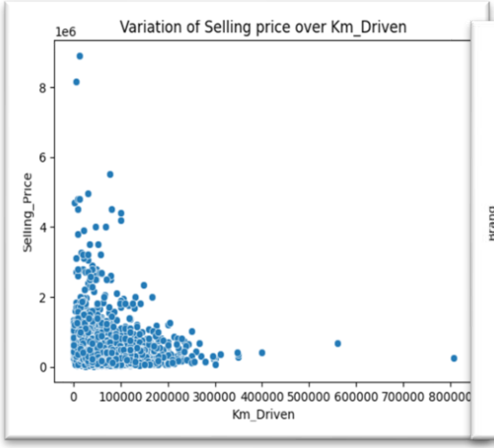


Graphical Analysis and conclusion on Data

Hist Plot on continuous data



Scatter Plot of Selling price over km, brand & Year



- The **Histplot** shows the distribution of continuous numerical.
- The various **Scatter plot** shows the variation of selling price indicating the impact of year(age), kms (use) ,brand (popularity) on the price of resale value of a car.

Data Cleaning & Pre-Processing Steps.

skill academy

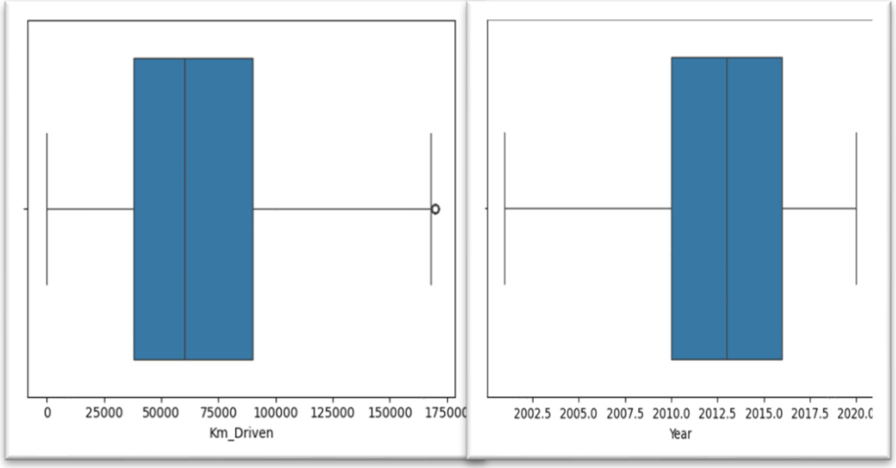
Few data cleaning steps are discussed in previous slides.

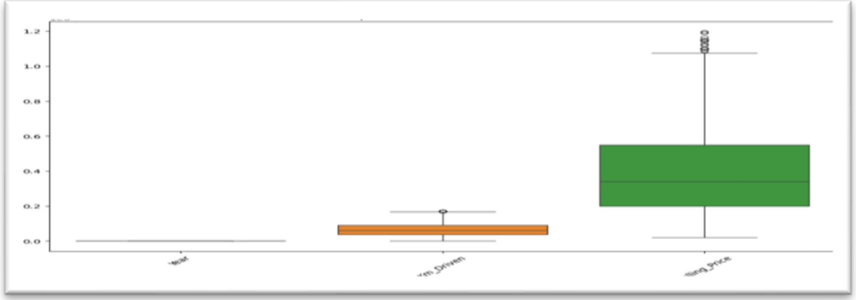
Treating Outliers (Data Capping)

```python
df_cap = df_changed
def capping_data(df_cap):
    for i in df_cap.columns:
        print("Check : Capped the", i, "Column")
        if df_cap[i].dtype in ["float64", "int64"]:
            Q1 = df_cap[i].quantile(0.25)
            Q3 = df_cap[i].quantile(0.75)
            IQR = Q3 - Q1
            df_cap = df_cap[
                ~((df_cap[i] < (Q1 - 1.5 * IQR)) |
                  (df_cap[i] > (Q3 + 1.5 * IQR)))
            ]
        else:
            df_cap[i] = df_cap[i]
    return df_cap
data = capping_data(df_cap)
```

- The above code caps the data value of numerical data columns using the IQR formulas.
- The scatter plot on the right shows the capped effect on outlier.
- Capping helps us to remove the data which may change the model prediction due to outlier data.

ScatterPlot (After Capping Data)





Data Cleaning & Pre-Processing Steps(In ML Model).

skill academy

Step 1 – Import Necessary Library and dataset

Module Imports

💡 Click here to ask Blackbox to help you code faster

```python
import pandas as pd                 # for data cleaning and data pre-processing, CSV file I/O,etc
import numpy as np                  # linear algebra & for mathematical computation
import matplotlib.pyplot as plt # for visualization
%matplotlib inline
import seaborn as sns               # for visualization
from collections import Counter # to count occurrences
from tabulate import tabulate   # to make tables for results

import warnings                     # for warning removals in code output
warnings.filterwarnings('ignore')

# Scalers & Encoders
from sklearn.preprocessing import StandardScaler, LabelEncoder
#train-test split
from sklearn.model_selection import train_test_split
# Metrics
from sklearn.metrics import (mean_squared_error, r2_score)
# Model Libraries
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor

import pickle          #to save and load model files as pkl file
```

[74]

- Import the basic library for data handling & the model handling ones from sklearn library.
- The next step is similar to previous to import the dataset but we will import the processed dataset instead of raw which we handled in previous steps in graphical analysis
- To save the dataset in last file at end we did - ***data.to_csv(“<file-path/file-name>.csv", index=False)***

Step 2 – Encoding dataset

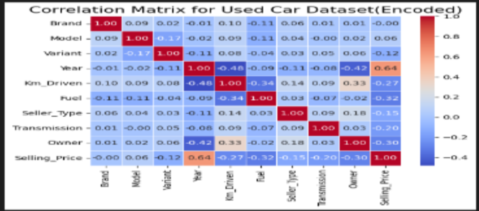
💡 Click here to ask Blackbox to help you code faster

```python
# Using Label Encoder to encode categorical data
label_encoder = LabelEncoder()  # instance of encoder
# Loop to  encode data in df
for feature in category_col:
    df_one[feature] = label_encoder.fit_transform(df[feature])
df_one.head()
```

	Brand	Model	Variant	Year	Km_Driven	Fuel	Seller_Type	Transmission	Owner	Selling_Price
0	12	5	546	2007	70000	4	1	1	0	60000
1	12	547	657	2007	50000	4	1	1	0	135000
2	10	517	311	2012	100000	1	1	1	0	600000
3	5	369	659	2017	46000	4	1	1	0	250000
4	9	35	729	2014	141000	1	1	1	2	450000

💡 Click here to ask Blackbox to help you code faster

```python
corr = df_one.corr()
sns.heatmap(corr, cmap="coolwarm", annot=True, fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix for Used Car Dataset(Encoded)", fontsize=16)
plt.savefig("../src/visualization/Correlation Graph - After Label Encoding")
plt.show()
```



Saving the encoded dataset

💡 Click here to ask Blackbox to help you code faster

```python
cleaned_data = df_one.to_csv("../data/processed/cleaned_data.csv", index=False)
```

- The above code encodes the loaded dataset using label encoder so that we can use the categorical values in model training as it needs number to build model.
- The correlation matrix shows us that the new encoded values are in numeric.

Data Cleaning & Pre-Processing Steps(In ML Model).

skill academy

Step 3 – Splitting data into Dependant & Independent variables for train-test split

Dependant (y) & Independent (x) features

1. Dropping dependant feature from dataset

+ Code + Markdown

Click here to ask Blackbox to help you code faster

```
x = df_one.drop(["Selling_Price"], axis=1)
y = df_one["Selling_Price"]

print(type(x))
print(type(y))
print(x.shape)
print(y.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
(3250, 9)
(3250,)
```

2. Splitting The dataset

Click here to ask Blackbox to help you code faster

```
# splitting the dataset into 70% training data and 30% test data
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.3, random_state=42
)
print(f"Split Check Test values : {3194 * 0.3} & Train values : {3194 * 0.7}")
# rows , columns
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
Split Check Test values : 958.1999999999999 & Train values : 2235.7999999999997
(2275, 9)
(975, 9)
(2275,)
(975,)
```

4. Standarizing the dataset

Click here to ask Blackbox to help you code faster

```
sc = StandardScaler()
sc.fit_transform(X_train)
sc.transform(X_test)
```

```
array([[-0.00522286,  0.38325919, -0.213698  , ...,  0.44793592,
         0.26187172, -0.71016833],
       [ 1.65825717, -0.23064065,  0.93736498, ...,  0.44793592,
         0.26187172,  2.45929306],
       [-0.48050286,  1.12202001, -0.83066776, ...,  0.44793592,
         0.26187172, -0.71016833]
```

- The code shows us the basic step of model building where in we split the dataset into training data & testing data.
- This step is very important as without doing the split if we train the model it will know the results and if unknown values are introduced it might not function as intended.
- The practice of splitting data helps us to find various parameters such as accuracy, precision, recall values, r2 score, mse values etc which helps us to determine if our model is fit or not or should we tune it more or use some other modelling technique.
- After these step we move on to actual model building & selection step.

ML Modeling

I Will add the code in loop as writing the same piece of code repeatedly changing variable names and remembering them will be not good idea so instead I will do looping.

```python
# Create a list of models
models = [
    LinearRegression(),
    Ridge(),
    Lasso(),
    RandomForestRegressor(),
    KNeighborsRegressor(),
    DecisionTreeRegressor(),
    GradientBoostingRegressor(),
    AdaBoostRegressor(),
]

# Define a list of model names
model_names = [
    "Linear Regression",
    "Ridge Regression",
    "Lasso Regression",
    "Random Forest",
    "K-Nearest Neighbors",
    "Decision Tree",
    "Gradient Boosting",
    "Ada Boost",
]

# Initialize variables to keep track of the best model
best_model_name = None
best_r2_score = -float("inf")
best_model = None

# Initialize an empty list to store results
all_results = []

# Function to evaluate regression models


def evaluate_regression_model(model, model_name, X_test, y_test):
    y_pred = model.predict(X_test)

    # Mean Squared Error and R-squared Score
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Display results in tabular format
    results_table = [
        ["Model", model_name],
        ["Mean Squared Error", mse],
        ["R-squared Score", r2],
    ]

    print(tabulate(results_table, headers=[
          "Metric", "Value"], tablefmt="heavy_grid"))

    # Store results in the dictionary
    return {
        "Model": model_name,
        "Mean Squared Error": mse,
        "R-squared Score": r2,
    }

```

```python
# Iterate over the models
for model, model_name in zip(models, model_names):
    print(f"\n{('-' * 40)}\n{model_name}\n{('-' * 40)}")

    # Train the model
    model.fit(X_train, y_train)

    # Evaluate and store results for regression model
    results = evaluate_regression_model(model, model_name, X_test, y_test)
    all_results.append(results)

    # Update the best model if needed
    if results["R-squared Score"] > best_r2_score:
        best_r2_score = results["R-squared Score"]
        best_model_name = model_name
        best_model = model

# Create a DataFrame from the results
results_df = pd.DataFrame(all_results)

# Sorting the dataframe by 'R-squared Score' in descending order
sorted_results_df = results_df.sort_values(
    by="R-squared Score", ascending=False)

# Displaying the sorted dataframe
print("\nSorted Models by R-squared Score:")
print(sorted_results_df)

# Print the best model based on R-squared score
print(f"\nBest Model based on R-squared Score: {best_model_name}")
best_model_name
```

ML Modeling

skill academy

In the previous slide we saw the code for model building – scoring ,parameters , function to check them on various passed models , save the results and iterate to next model. Here we see few glimpse of the code o/p.

Linear Regression

Metric	Value
Model	Linear Regression
Mean Squared Error	28427461410.760574
R-squared Score	0.5370094957944969

Ridge Regression

Metric	Value
Model	Ridge Regression
Mean Squared Error	28429012702.72386
R-squared Score	0.53698423031483

Lasso Regression

Metric	Value
Model	Lasso Regression
Mean Squared Error	28427489885.780876
R-squared Score	0.5370090320294136

Random Forest

Metric	Value
Model	Random Forest
Mean Squared Error	13871188171.954166
R-squared Score	0.774083647045899

k-Nearest Neighbors

Metric	Value
Model	k-Nearest Neighbors
Mean Squared Error	54114480862.42737
R-squared Score	0.11865184100362591

Decision Tree

Metric	Value
Model	Decision Tree
Mean Squared Error	26104259077.00128
R-squared Score	0.5748468743889662

Gradient Boosting

Metric	Value
Model	Gradient Boosting
Mean Squared Error	14369733662.247465
R-squared Score	0.7659639692250402

Ada Boost

Metric	Value
Model	Ada Boost
Mean Squared Error	29315494521.837402
R-squared Score	0.522546336671431

Sorted Models by R-squared Score:

	Model	Mean Squared Error	R-squared Score
3	Random Forest	1.387119e+10	0.774084
6	Gradient Boosting	1.436973e+10	0.765964
5	Decision Tree	2.610426e+10	0.574847
0	Linear Regression	2.842746e+10	0.537009
2	Lasso Regression	2.842749e+10	0.537009
1	Ridge Regression	2.842901e+10	0.536984
7	Ada Boost	2.931549e+10	0.522546
4	k-Nearest Neighbors	5.411448e+10	0.118652

Best Model based on R-squared Score: Random Forest
'Random Forest'

Model performance table (Basic)

```
model_performance = pd.DataFrame(results_df)
model_performance
```

	Model	Mean Squared Error	R-squared Score
0	Linear Regression	2.842746e+10	0.537009
1	Ridge Regression	2.842901e+10	0.536984
2	Lasso Regression	2.842749e+10	0.537009
3	Random Forest	1.387119e+10	0.774084
4	k-Nearest Neighbors	5.411448e+10	0.118652
5	Decision Tree	2.610426e+10	0.574847
6	Gradient Boosting	1.436973e+10	0.765964
7	Ada Boost	2.931549e+10	0.522546

ML Modeling

skill academy

In the previous slide we saw the model scores o/p and best model the next step is to save the model(will skip ss can look into notebook) now we see the testing part of model against test data that we split.

Generating sample data from cleaned df to test on the trained model.

```python
random_datasample = df_one.sample(20)
random_datasample_df = random_datasample.drop("Selling_Price", axis=1)
print(random_datasample_df.shape)
random_datasample_df.head()
```

```
(20, 9)
```

	Brand	Model	Variant	Year	Km_Driven	Fuel	Seller_Type	Transmission	Owner
2308	12	588	37	2008	80000	4	1	1	1

Making Predictions on sample dataset against the trained model

```python
# making prediction on random data
predicted_data = load_model.predict(testsample_df)
print(f"The predicted data from {f_modelname} model:\n", predicted_data)
```

```
The predicted data from Random Forest model:
 [126650.         132452.66666667 572970.         523030.
```

Comparision of Actual and Predicted values by the model

```python
# Compare the actual data and predicted data
prediction_data = random_datasample.copy()
prediction_data["predicted_target"] = predicted_data
# Calculate the absolute percentage difference
prediction_data["percentage_difference"] = abs(
    (random_datasample["Selling_Price"] - predicted_data) /
    random_datasample["Selling_Price"]) * 100
# Print the actual and predicted data
print(f"Actual Data and Predicted Data Comparison based on {
      f_modelname} model:\n")
# Display the results where the absolute percentage difference is less than or equal to 20%
safe_predictions = prediction_data[prediction_data["percentage_difference"] <= 20]
# Print the safe predictions
print("Safe Predictions:")
print(safe_predictions[["Selling_Price",
      "predicted_target", "percentage_difference"]])
# Print the count and percentage of safe predictions
safe_percentage = (len(safe_predictions) / len(prediction_data)) * 100
print(f"\nPercentage of Safe Predictions: {safe_percentage:.2f}%")
if safe_percentage >= 90:
    print(
        f"\nOur model based on '{f_modelname}' is well trained, with {
            safe_percentage:.2f}% safe predictions."
    )
else:
    print(f"Our model based on '{f_modelname}' needs more training to improve safety, currently at {
          safe_percentage:.2f}% safe predictions.")
# Save the results as a DataFrame
final_results_df = prediction_data[[
    "Selling_Price", "predicted_target", "percentage_difference"]]
final_results_df.to_csv('../reports/model_predicted_results.csv', index=False)
```

```
Actual Data and Predicted Data Comparison based on Random Forest model:

Safe Predictions:
      Selling_Price  predicted_target  percentage_difference
2308         130000     126650.000000               2.576923
379          500000     572970.000000              14.594000
3116         535000     523030.000000               2.237383
2143         490000     509029.940000               3.883661
1971         836000     768859.980000               8.031103
1509         530000     504139.990000               4.879247
2672         280000     266050.000000               4.982143
692          400000     369299.980000               7.675005
1268          70000      73140.000000               4.485714
1014         430000     410189.970000               4.606984
480          535000     633740.000000              18.456075
2279         280000     238370.000000              14.867857
389          320000     318026.666667               0.616667
2800         280000     295100.000000               5.392857
720          940000     951180.000000               1.189362
1613         190000     213750.000000              12.500000
1750         280000     267481.101111               4.471035

Percentage of Safe Predictions: 85.00%
Our model based on 'Random Forest' needs more training to improve safety, currently at 85.00% safe predictions.
```

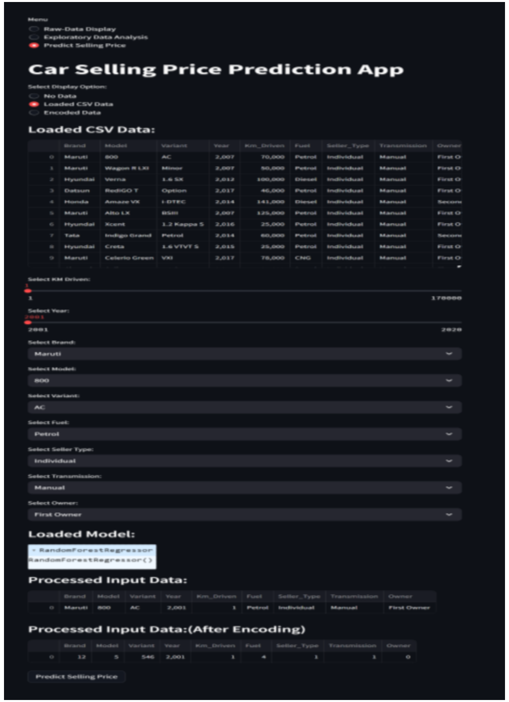
- The code above generates 20 random sample data from original file which we used for train test split and test them to predict value against model.
- The code on side checks the actual value and predicted value (I tweaked it a bit so that if difference of selling price is say under certain limit its safe prediction.

Deployment of ML Models using Streamlit.

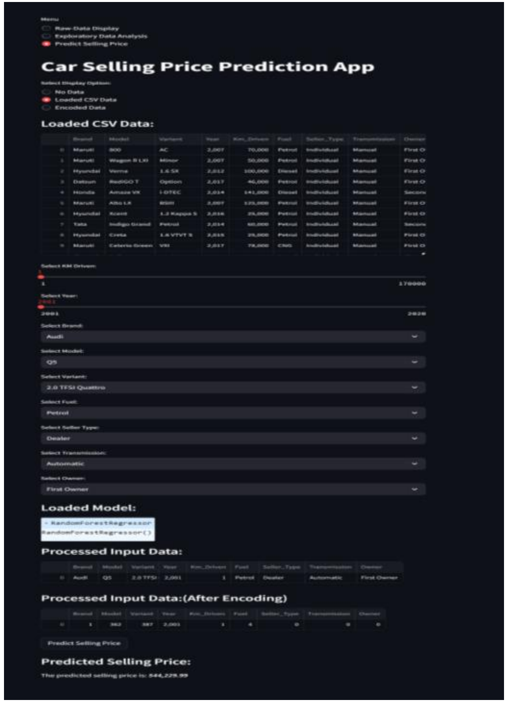
skill academy

Please visit the link to view the actual working code of the project (won’t paste long code)

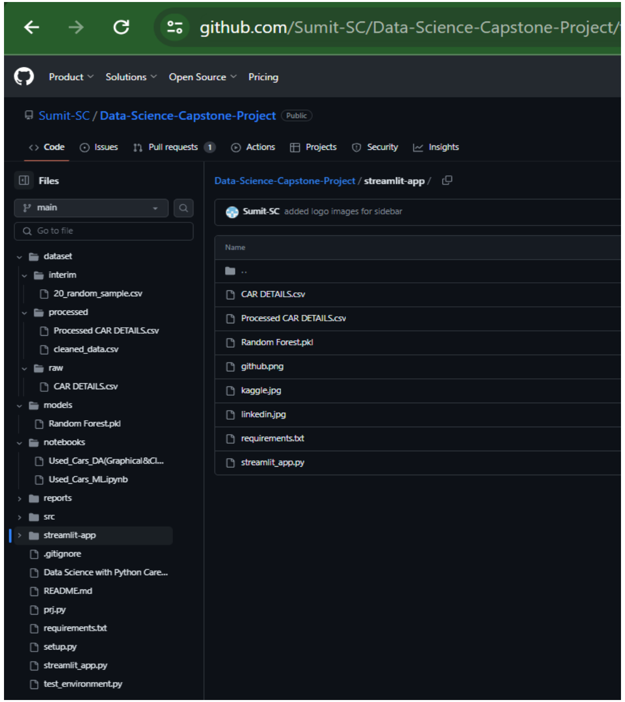
Without any selection



Prediction of Selling Price



Github Code Structure



Endnotes

skill academy

Reference Links:-

- GitHub Repo Link
- Streamlit App Weblink
- Streamlit App Alternate
- Google Drive Folder
- Zip file
- Google Colab Links (Run & View code Directly)
  - EDA & Graphical Analysis Colab
  - Model Training & Evaluation Colab
  - Readme File (To get the project structure & all links)

About Me :

- Name – Sumit S. Chaure
- Batch – 10
- Course – Data Science With Python
- GitHub - Sumit-SC
- Kaggle – Sumit
- Email - @

END