CAR SALES PREDICTION USING MACHINE LEARNING ALGORITHMS

A Project report submitted in partial fulfillment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

IN

DATA ANALYTICS

Submitted by

Mohd Zain   Admission No : 22MT0214

Under the guidance of

Mrs. Shuvashree Mondal
Assistant Professor

DEPARTMENT OF MATHEMATICS AND COMPUTING
INDIAN INSTITUTE OF TECHNOLOGY
(ISM) DHANBAD

ABSTRACT

Machine Learning is used across many ranges around the world.
Sales prediction is the current numero trend in which all the business companies thrive and it also aids the organization or concern in determining the future goals for it and its plan and procedure to achieve it.
In this project, the application of supervised machine learning techniques are used to predict the sales of cars. The predictions are based on historical data collected from Kaggle, an online verified repository. Different techniques like Linear regression, Adaboost Regression, random forest, KNN Regression, Support Vector Machines and decision trees have been used to make the predictions. I then evaluated and compared the predictions so that we are able to find the algorithms and techniques which provide the best performances. All methods provided comparable performance. In the future, we plan to use more techniques and methods.

Keywords: Linear Regression; Adaboost Regression; Decision Tree; Random Forest; KNeighbours Regression; SVM; python programming;
.

Introduction

The prediction of car sales is an important and rewarding problem in current times. According to data obtained from the Government of India , the revenue generated from cars registered between 2019-2020 and 2020-21 has witnessed a spectacular increase of 44%. The number of cars in 2019 has reached 55,634,824. With a rise in new technologies and advancements, it is likely that sale of cars and the scope of this study will see a growth.

Machine learning proves to be effective in assisting in making predictions from the large quantity of data produced by the car industries. This project aims to predict the car sales by analyzing data of sales.. Machine Learning techniques can be a boon in this regard. By collecting the data from various sources, using them under suitable headings & finally analysing to extract the desired data we can say that this technique can be very well adapted to do the prediction of car sales.

I used a multivariate regression model to predict the sales of more then 25 Car Manufacturing Companies. The results were satisfactory and can be repeated with accuracy. Support Vector Machines (SVM) were also used on the same data set to see the r2Score.

Different Machine Learning techniques were used to measure the variability in prediction score and then using the best or the most accurate techniques for future in this project.

DATASET

Data Analysis

Data analysis refers to the process of manipulating raw data to uncover useful insights and draw conclusions

In this project dataset in taken from recognised platform named Kaggle, "car_sales.csv".
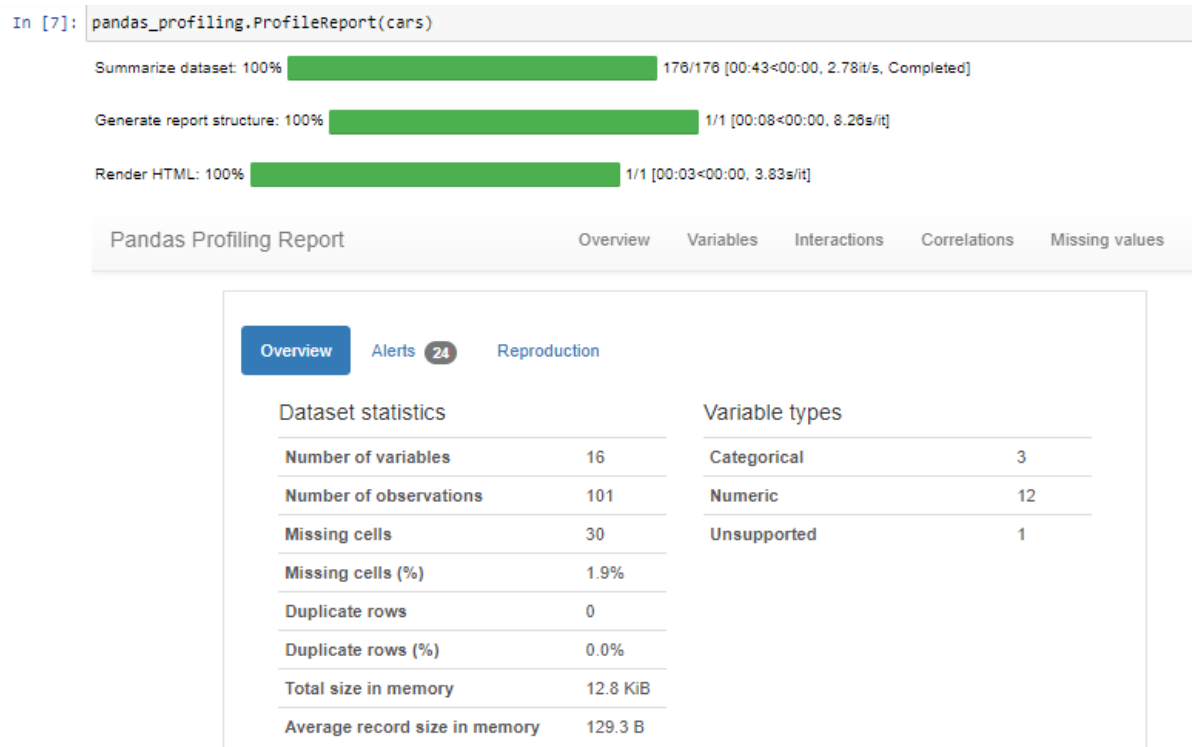
Dataset here contains some NAN values i.e. some missing values, there are 3 categorical features as well as 12 numeric features.
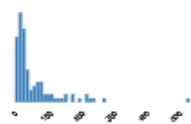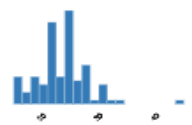
On describing dataset it shows:

```
cars.describe(include='all')
```

	Manufacturer	Model	Sales_in_thousands	__year_resale_value	Vehicle_type	Price_in_thousands	Engine_size	Horsepower	Wheelbase	Width
count	101	101	101.000000	85.000000	101	99.000000	100.000000	100.000000	100.000000	100.000000
unique	19	101	NaN	NaN	2	NaN	NaN	NaN	NaN	NaN
top	Ford	Integra	NaN	NaN	Passenger	NaN	NaN	NaN	NaN	NaN
freq	11	1	NaN	NaN	74	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	59.405574	18.221059	NaN	28.854737	3.278000	195.440000	108.805000	71.801000
std	NaN	NaN	77.964018	10.730210	NaN	15.066433	1.127756	60.300312	8.093549	3.596786
min	NaN	NaN	0.110000	5.160000	NaN	9.235000	1.000000	55.000000	93.100000	62.600000
25%	NaN	NaN	14.785000	11.525000	NaN	18.962500	2.500000	153.750000	104.000000	69.175000
50%	NaN	NaN	29.450000	14.795000	NaN	24.997000	3.100000	194.000000	108.000000	71.450000
75%	NaN	NaN	73.203000	20.525000	NaN	35.717500	4.000000	225.000000	113.000000	74.400000
max	NaN	NaN	540.561000	58.600000	NaN	85.500000	8.000000	450.000000	138.700000	79.900000

If we check dataset through Pandas Profiling we see that:

```
In [7]: pandas_profiling.ProfileReport(cars)
```

Summarize dataset: 100% ██████████ 176/176 [00:43<00:00, 2.78it/s, Completed]

Generate report structure: 100% ██████████ 1/1 [00:08<00:00, 8.26s/it]

Render HTML: 100% ██████████ 1/1 [00:03<00:00, 3.83s/it]

Pandas Profiling Report   Overview   Variables   Interactions   Correlations   Missing values

Overview   Alerts 24   Reproduction

Dataset statistics

Number of variables	16
Number of observations	101
Missing cells	30
Missing cells (%)	1.9%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	12.8 KiB
Average record size in memory	129.3 B

Variable types

Categorical	3
Numeric	12
Unsupported	1

Some of the features of dataset analysed using Pandas Profiling:

Manufacturer
Categorical

HIGH CORRELATION

Distinct	19
Distinct (%)	18.8%
Missing	0
Missing (%)	0.0%
Memory size	936.0 B

Ford	11
Dodge	11
Mercedes-B	9
Chevrolet	9
Mitsubishi	7
Other values (14)	54

Toggle details

Model
Categorical

HIGH CARDINALITY
UNIFORM
UNIQUE

Distinct	101
Distinct (%)	100.0%
Missing	0
Missing (%)	0.0%
Memory size	936.0 B

Integra	1
Sonata	1
LX470	1
LS400	1
GS400	1
Other values (96)	96

Toggle details

Sales_in_thou...
Real number ($\mathbb{R}_{\geq 0}$)

HIGH CORRELATION
UNIQUE

Distinct	101
Distinct (%)	100.0%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	59.40557426

Minimum	0.11
Maximum	540.561
Zeros	0
Zeros (%)	0.0%
Negative	0
Negative (%)	0.0%
Memory size	936.0 B

Toggle details

Fuel_efficiency
Real number ($\mathbb{R}_{\geq 0}$)

HIGH CORRELATION
MISSING

Distinct	20
Distinct (%)	20.2%
Missing	2
Missing (%)	2.0%
Infinite	0
Infinite (%)	0.0%
Mean	23.3030303

Minimum	15
Maximum	45
Zeros	0
Zeros (%)	0.0%
Negative	0
Negative (%)	0.0%
Memory size	936.0 B

Toggle details

Latest_Launch
Unsupported

REJECTED
UNSUPPORTED

Missing	0
Missing (%)	0.0%
Memory size	936.0 B

Summary of Continous and Categorical features of the dataset.

Each feature represents a measurable piece of data that canbe used for analysis.

```python
def continuous_var_summary(x):
    return pd.Series([x.count(), x.isnull().sum(), x.sum(), x.mean(), x.median(),
                     x.std(), x.var(), x.min(), x.quantile(0.01), x.quantile(0.05),
                         x.quantile(0.10),x.quantile(0.25),x.quantile(0.50),x.quantile(0.75),
                             x.quantile(0.90),x.quantile(0.95), x.quantile(0.99),x.max()],
                index = ['N', 'NMISS', 'SUM', 'MEAN','MEDIAN', 'STD', 'VAR', 'MIN', 'P1',
                            'P5' ,'P10' ,'P25' ,'P50' ,'P75' ,'P90' ,'P95' ,'P99' ,'MAX'])
```

```python
def categorical_var_summary(x):
    Mode = x.value_counts().sort_values(ascending = False)[0:1].reset_index()
    return pd.Series([x.count(), x.isnull().sum(), Mode.iloc[0, 0], Mode.iloc[0, 1],
                        round(Mode.iloc[0, 1] * 100/x.count(), 2)],
                index = ['N', 'NMISS', 'MODE', 'FREQ', 'PERCENT'])
```

Continous Feature

```python
cars_conti_vars.apply(continuous_var_summary).T.round(2)
```

	N	NMISS	SUM	MEAN	MEDIAN	STD	VAR	MIN	P1	P5	P10	P25	P50	P75	P90	P95	P99
Sales_in_thousands	101.0	0.0	5999.96	59.41	29.45	77.96	6078.39	0.11	0.92	3.31	6.54	14.78	29.45	73.20	155.79	220.65	276.75
__year_resale_value	85.0	16.0	1548.79	18.22	14.80	10.73	115.14	5.16	5.75	7.83	8.86	11.52	14.80	20.52	31.14	40.10	58.49
Price_in_thousands	99.0	2.0	2856.62	28.85	25.00	15.07	227.00	9.24	9.69	12.29	13.98	18.96	25.00	35.72	45.81	60.29	82.66
Engine_size	100.0	1.0	327.80	3.28	3.10	1.13	1.27	1.00	1.50	1.80	2.00	2.50	3.10	4.00	4.60	5.20	5.72
Horsepower	100.0	1.0	19544.00	195.44	194.00	60.30	3636.13	55.00	91.63	112.85	120.00	153.75	194.00	225.00	275.00	300.10	346.05
Wheelbase	100.0	1.0	10880.50	108.80	108.00	8.09	65.51	93.10	93.40	96.20	98.94	104.00	108.00	113.00	117.52	120.74	138.50
Width	100.0	1.0	7180.10	71.80	71.45	3.60	12.94	62.60	65.67	66.70	67.28	69.18	71.45	74.40	76.82	78.70	79.31
Length	100.0	1.0	18976.70	189.77	190.55	14.15	200.17	149.40	151.97	167.46	174.39	180.00	190.55	199.77	207.25	212.00	224.20
Curb_weight	99.0	2.0	345.29	3.49	3.47	0.66	0.44	1.90	2.23	2.46	2.67	3.06	3.47	3.88	4.25	4.47	5.40
Fuel_capacity	100.0	1.0	1859.10	18.59	18.00	4.13	17.06	10.30	11.88	13.18	14.27	16.00	18.00	20.00	24.37	26.00	32.00
Fuel_efficiency	99.0	2.0	2307.00	23.30	23.00	4.51	20.34	15.00	15.00	16.00	17.80	21.00	23.00	25.50	28.00	30.00	33.24
Power_perf_factor	99.0	2.0	8019.58	81.01	80.66	26.66	710.71	23.28	36.40	45.66	49.45	62.47	80.66	92.65	114.21	126.21	142.08

Categorical Feature

```python
cars_cat_vars.apply(categorical_var_summary).T
```

	N	NMISS	MODE	FREQ	PERCENT
Manufacturer	101	0	Ford	11	10.89
Model	101	0	Integra	1	0.99
Vehicle_type	101	0	Passenger	74	73.27
Latest_Launch	101	0	2/23/2012	2	1.98

Further steps to analyse data:

Cleaning of data: is the process of uncovering and correcting, or eliminating inaccurate or repeat records from your dataset. During the data wrangling process, we'll transform the raw data into a more useful format.

1. *Outlier Treatment*: an outlier is an extremely high or extremely low data point relative to the nearest data point and the rest of the neighbouring.

Outlier Treatment

```
cars_conti_vars = cars_conti_vars.apply(lambda x: x.clip(lower = x.quantile(0.01), upper = x.quantile(0.99)))
cars_conti_vars.apply(continuous_var_summary).T.round(2)
```

	N	NMISS	SUM	MEAN	MEDIAN	STD	VAR	MIN	P1	P5	P10	P25	P50	P75	P90	P95	P99	
Sales_in_thousands	101.0	0.0	5736.95	56.80	29.45	65.02	4227.86	0.92	0.92	3.31	6.54	14.78	29.45	73.20	155.79	220.65	276.75	2
__year_resale_value	85.0	16.0	1549.27	18.23	14.80	10.72	114.85	5.75	5.84	7.83	8.86	11.52	14.80	20.52	31.14	40.10	58.47	
Price_in_thousands	99.0	2.0	2854.23	28.83	25.00	14.95	223.61	9.69	9.70	12.29	13.98	18.96	25.00	35.72	45.81	60.29	82.60	
Engine_size	100.0	1.0	326.02	3.26	3.10	1.04	1.09	1.50	1.50	1.80	2.00	2.50	3.10	4.00	4.60	5.20	5.70	
Horsepower	100.0	1.0	19476.68	194.77	194.00	55.86	3119.87	91.63	92.00	112.85	120.00	153.75	194.00	225.00	275.00	300.10	345.01	3
Wheelbase	100.0	1.0	10880.60	108.81	108.00	8.08	65.29	93.40	93.40	96.20	98.94	104.00	108.00	113.00	117.52	120.74	138.50	1
Width	100.0	1.0	7182.58	71.83	71.45	3.52	12.37	65.67	65.70	66.70	67.28	69.18	71.45	74.40	76.82	78.70	79.30	
Length	100.0	1.0	18978.98	189.79	190.55	14.07	197.93	151.97	152.00	167.46	174.39	180.00	190.55	199.77	207.25	212.00	224.20	2
Curb_weight	99.0	2.0	345.46	3.49	3.47	0.65	0.42	2.23	2.24	2.46	2.67	3.06	3.47	3.88	4.25	4.47	5.40	
Fuel_capacity	100.0	1.0	1860.68	18.61	18.00	4.10	16.82	11.88	11.90	13.18	14.27	16.00	18.00	20.00	24.37	26.00	32.00	
Fuel_efficiency	99.0	2.0	2295.24	23.18	23.00	4.07	16.53	15.00	15.00	16.00	17.80	21.00	23.00	25.50	28.00	30.00	33.00	
Power_perf_factor	99.0	2.0	7986.65	80.67	80.66	24.86	617.82	36.40	36.67	45.66	49.45	62.47	80.66	92.65	114.21	126.21	141.16	1

2. *Missing Value Treatment*: occur when no data value is stored for the variable in ana observation. Missing data are a common occurrence and can have a significant effects on the conclusions that can be drawn from thr data.

```
# Missing value imputation for categorical and continuous variables
def missing_imputation(x, stats = 'mean'):
    if (x.dtypes == 'float64') | (x.dtypes == 'int64'):
        x = x.fillna(x.mean()) if stats == 'mean' else x.fillna(x.median())
    else:
        x = x.fillna(x.mode())
    return x
```

Missing Value Treatment

```
cars_conti_vars = cars_conti_vars.apply(missing_imputation)
cars_cat_vars = cars_cat_vars.apply(missing_imputation)
cars_conti_vars.apply(continuous_var_summary).T.round(1)
```

```
C:\Users\HTC\anaconda3\lib\site-packages\pandas\core\algorithms.py:968: UserWarning: Unable to sort modes: '<' not support
tween instances of 'str' and 'datetime.datetime'
  warn(f"Unable to sort modes: {err}")
```

	N	NMISS	SUM	MEAN	MEDIAN	STD	VAR	MIN	P1	P5	P10	P25	P50	P75	P90	P95	P99	MAX
Sales_in_thousands	101.0	0.0	5737.0	56.8	29.4	65.0	4227.9	0.9	0.9	3.3	6.5	14.8	29.4	73.2	155.8	220.6	276.7	276.7
__year_resale_value	101.0	0.0	1840.9	18.2	17.3	9.8	96.5	5.7	5.9	7.8	9.1	12.5	17.3	19.7	28.7	39.0	58.5	58.5
Price_in_thousands	101.0	0.0	2911.9	28.8	25.3	14.8	219.1	9.7	9.7	12.3	14.0	19.0	25.3	35.3	45.7	60.1	82.6	82.7
Engine_size	101.0	0.0	329.3	3.3	3.1	1.0	1.1	1.5	1.5	1.8	2.0	2.5	3.1	4.0	4.6	5.2	5.7	5.7
Horsepower	101.0	0.0	19671.4	194.8	194.8	55.6	3088.7	91.6	92.0	113.0	120.0	154.0	194.8	225.0	275.0	300.0	345.0	346.1
Wheelbase	101.0	0.0	10989.4	108.8	108.0	8.0	64.6	93.4	93.4	96.2	99.0	104.1	108.0	113.0	117.5	120.7	138.5	138.5
Width	101.0	0.0	7254.4	71.8	71.5	3.5	12.2	65.7	65.7	66.7	67.3	69.2	71.5	74.4	76.8	78.7	79.3	79.3
Length	101.0	0.0	19168.8	189.8	190.4	14.0	196.0	152.0	152.0	167.5	174.4	180.1	190.4	199.7	207.2	212.0	224.2	224.2
Curb_weight	101.0	0.0	352.4	3.5	3.5	0.6	0.4	2.2	2.2	2.5	2.7	3.1	3.5	3.9	4.2	4.5	5.4	5.4
Fuel_capacity	101.0	0.0	1879.3	18.6	18.0	4.1	16.7	11.9	11.9	13.2	14.3	16.0	18.0	20.0	24.3	26.0	32.0	32.0
Fuel_efficiency	101.0	0.0	2341.6	23.2	23.2	4.0	16.2	15.0	15.0	16.0	18.0	21.0	23.2	25.0	28.0	30.0	33.0	33.2
Power_perf_factor	101.0	0.0	8148.0	80.7	80.7	24.6	605.5	36.4	36.7	45.8	49.6	62.5	80.7	92.4	113.9	125.3	141.1	142.1

3. *Handling Categorical Values:* categorical data has values and observations which can be sorted into categories or groups like Gender.

```python
# An utility function to create dummy variable
def create_dummies(df, colname):
    col_dummies = pd.get_dummies(df[colname], prefix = colname, drop_first = True)
    df = pd.concat([df, col_dummies], axis = 1)
    df.drop(colname, axis = 1, inplace = True )
    return df

cars_cat_vars = cars[['Manufacturer', 'Vehicle_type']]

for c_feature in ['Manufacturer', 'Vehicle_type']:
    cars_cat_vars[c_feature] = cars_cat_vars[c_feature].astype('category')
    cars_cat_vars = create_dummies(cars_cat_vars, c_feature)
```

Final data: contains dummy variables for categorical features.

Final Data

```python
cars_new = pd.concat([cars_conti_vars, cars_cat_vars], axis = 1)
cars_new.head(5)
```

	Sales_in_thousands	__year_resale_value	Price_in_thousands	Engine_size	Horsepower	...	Manufacturer_Lincoln	Manufacturer_Mercedes-B
0	16.919	16.360	21.500000	1.8	140.0	...	0	0
1	39.384	19.875	28.400000	3.2	225.0	...	0	0
2	14.114	18.225	28.830623	3.2	225.0	...	0	0
3	8.588	29.725	42.000000	3.5	210.0	...	0	0
4	20.397	22.255	23.990000	1.8	150.0	...	0	0

5 rows × 31 columns

Checking Data is normally Distributed or not because Normal distribution accurately describes the distribution of values for many natural phenomena. On checking we find that target variable is not normally distributed so we apply log transformation: log recalls the data and make the distribution normal.
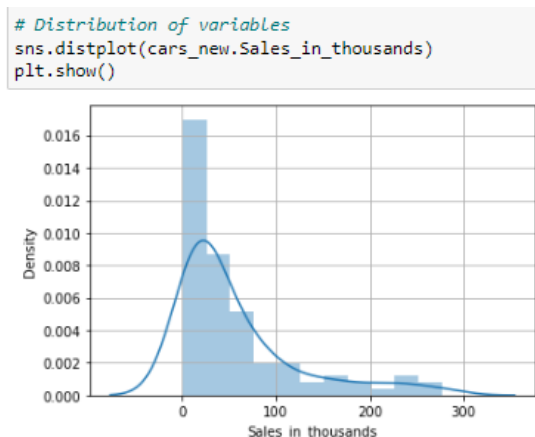
```python
# Distribution of variables
sns.distplot(cars_new.Sales_in_thousands)
plt.show()
```

```python
# apply log transformation: log is rescalling the data and making the distribution normal
cars_new['ln_sales_in_thousands'] = np.log(cars_new['Sales_in_thousands']+1)

# Distribution of variables
sns.distplot(cars_new.ln_sales_in_thousands)
plt.show()
```
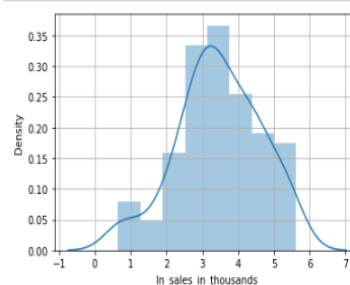

Fig1. Target feature Distribution   Fig2. Target feature distribution after log.

Scaling of dataset:

Scaling of features is an essential step in modelling the algorithms with the datsets. The data obtained contains features of various dimensions and scales altogether. Different scales of the data features affect the modeling of a dataset adversely.

It leads to a biased outcome of predictions in terms of misclassification error and accuracy rates. Thus, it is necessary to Scale the data prior to modeling. Standardization is a scaling technique wherein it makes the data scale-free by converting the statistical distribution of the data into the below format:

mean - 0 (zero)

standard deviation - 1

For Standardization we use StandardScaler Function

```
from sklearn.preprocessing import StandardScaler
standard = StandardScaler()
x = cars_new.loc[:, feature_columns].values
scaled_data = standard.fit_transform(x)
scaled_data
```

```
array([[-1.33321966, -1.41501514, -1.33146355, ...,  0.60404045,
        -0.95074819, -1.29985877],
       [ 0.04309155, -0.05831857, -0.34644195, ...,  0.60404045,
        -0.0882487 , -0.43821676],
       [-0.03058343, -0.05831857, -0.34644195, ...,  0.60404045,
        -0.23824861, -0.35205256],
```

Training and Testing of Dataset:

Training data is the subset of original data that is used to train the machine learning model, whereas testing data is used to check the accuracy of the model. The training dataset is generally larger in size compared to the testing dataset.

```
train_X, test_X, train_y, test_y = train_test_split(scaled_data,
                                    cars_new['ln_sales_in_thousands'], test_size = 0.2, random_state = 1000)
```

TECHNIQUES USED IN PROJECT

Linear Regression:

Linear regression is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables.

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.

Linear regression was the first type of regression analysis to be studied rigorously, and to be used extensively in practical applications.[4] This is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and because the statistical properties of the resulting estimators are easier to determine.

```python
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
LR.fit(train_X,train_y)
```

```
LinearRegression()
```

```python
y_pred =  LR.predict(test_X)
```

```python
mae=mean_absolute_error(test_y,y_pred)
mse=mean_squared_error(test_y,y_pred)
r2=r2_score(test_y,y_pred)
print("Mean Absolute Error:",mae)
print("Mean Squared Error:",mse)
print("R2 Score:",r2)
```

```
Mean Absolute Error: 0.6838780179007596
Mean Squared Error: 0.6670703218994839
R2 Score: 0.6630602803388224
```

Adaboost Regression

An AdaBoost regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases.

The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction.

```
from sklearn.ensemble import AdaBoostRegressor
model=AdaBoostRegressor(n_estimators=40,random_state=4)
model.fit(train_X,train_y)
```

```
AdaBoostRegressor(n_estimators=40, random_state=4)
```

```
y_predAB=model.predict(test_X)
print("R2 Score Using Adaboost",r2_score(test_y,y_predAB))
```

```
R2 Score Using Adaboost 0.6104852873692055
```

Random Forest Regression:

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
In random forests each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.
The purpose of these two sources of randomness is to decrease the variance of the forest estimator.

```
from sklearn.ensemble import RandomForestRegressor
model=RandomForestRegressor(n_estimators=15,random_state=4)
model.fit(train_X,train_y)
```

```
RandomForestRegressor(n_estimators=15, random_state=4)
```

```
y_predRF=model.predict(test_X)
r2=r2_score(test_y,y_predRF)
print(r2)
```

```
0.6586800762961482
```

KNN Regression:

KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighbourhood.The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.

```python
from sklearn import neighbors
r2_val={}
for k in range(50):
    k=k+1
    model=neighbors.KNeighborsRegressor(n_neighbors=k)
    model.fit(train_X,train_y)
    y_predKNN=model.predict(test_X)
    r2=r2_score(test_y,y_predKNN)
    r2_val[k]=r2

v=list(r2_val.values())
k=list(r2_val.keys())
print("max r2 score: ",max(v))
```

```
max r2 score:  0.5582305706969565
```

SVM Regression:

The method of Support Vector Classification can be extended to solve regression problems. This method is called Support Vector Regression.
The model produced by support vector classification depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by Support Vector Regression depends only on a subset of the training data, because the cost function ignores samples whose prediction is close to their target.

```python
from sklearn.svm import SVR
model=SVR(kernel='rbf')
model.fit(train_X,train_y.ravel())
```

```
SVR()
```

```python
y_predSVM=model.predict(test_X)
r2=r2_score(test_y,y_predSVM)
print("R2 Score in SVR",r2)
```

```
R2 Score in SVR 0.4910363949268538
```

Decisition Tree Regression:

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

```
from sklearn.tree import DecisionTreeRegressor
dtree=DecisionTreeRegressor(ccp_alpha=0.02,random_state=30)
dtree.fit(train_X,train_y)
y_predDT=dtree.predict(test_X)
print("Decision Tree r2 score")
r2_score(test_y,y_predDT)
```

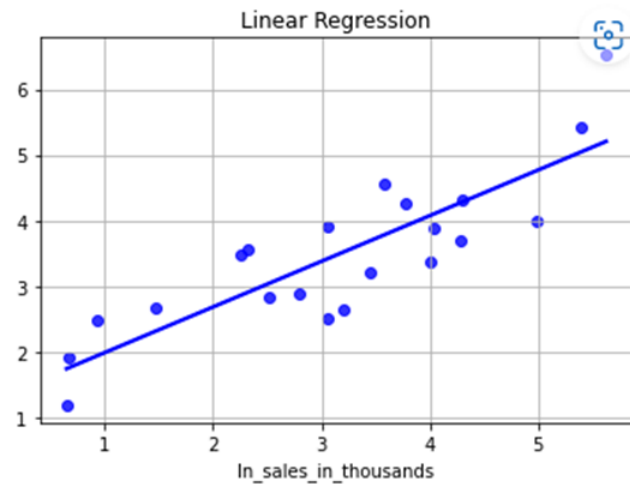Decision Tree r2 score

: 0.4999986853118058
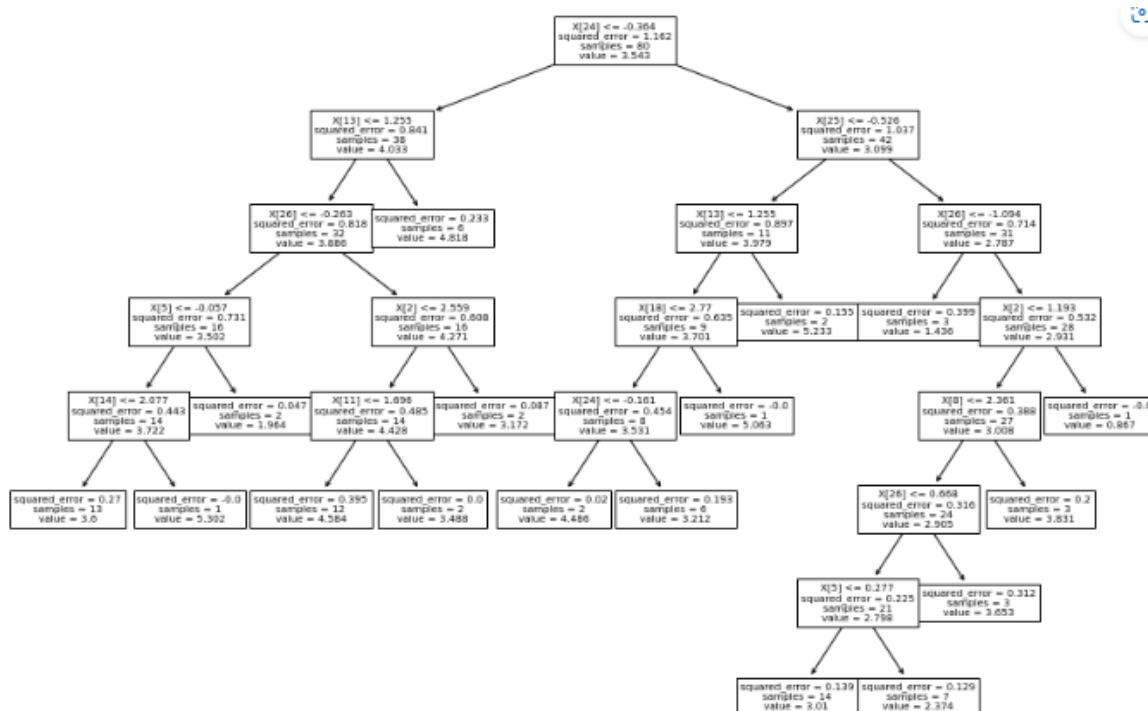
RESULTS AND PLOTS

## Linear Regression:

R2 Score: 0.66306

	Actual Value	Predicted Value
37	4.279246	3.697881
97	0.926637	2.481541
56	5.626711	6.523986
55	5.401100	5.412050
33	3.997834	3.362431
84	2.512603	2.840881
3	2.260512	3.481335
4	3.063251	2.512318
72	1.466491	2.667088
59	4.306805	4.311010

Adaboost Regression:

R2 Score: 0.61048

	Actual Value	Predicted Value
37	4.279246	3.240416
97	0.926637	2.782364
56	5.626711	4.741314
55	5.401100	4.860065
33	3.997834	3.659949
84	2.512603	3.054022
3	2.260512	3.237238
4	3.063251	3.160330
72	1.466491	2.196221
59	4.306805	3.488545
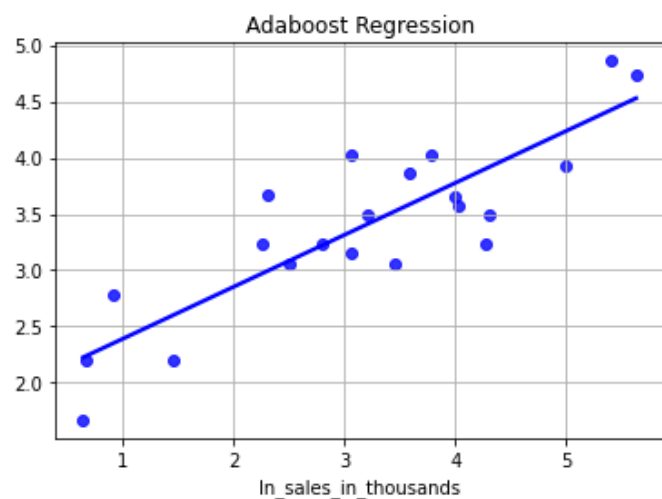
Random Forest Regression:

R2 Score: 0.65868

	Actual Value	Predicted Value
37	4.279246	3.359184
97	0.926637	2.493229
56	5.626711	4.634086
55	5.401100	4.912153
33	3.997834	3.639119
84	2.512603	3.234227
3	2.260512	3.060825
4	3.063251	3.233639
72	1.466491	2.392597
59	4.306805	4.248885
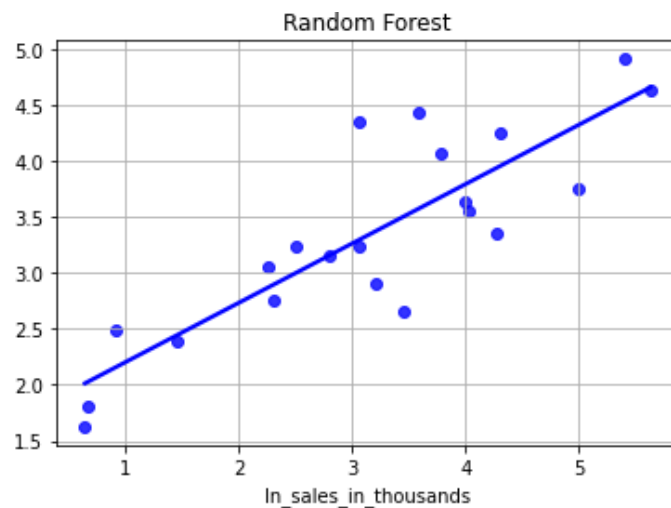
KNN Regression:

R2 Score: 0.55823057

	Actual Value	Predicted Value
37	4.279246	3.359184
97	0.926637	2.493229
56	5.626711	4.634086
55	5.401100	4.912153
33	3.997834	3.639119
84	2.512603	3.234227
3	2.260512	3.060825
4	3.063251	3.233639
72	1.466491	2.392597
59	4.306805	4.248885
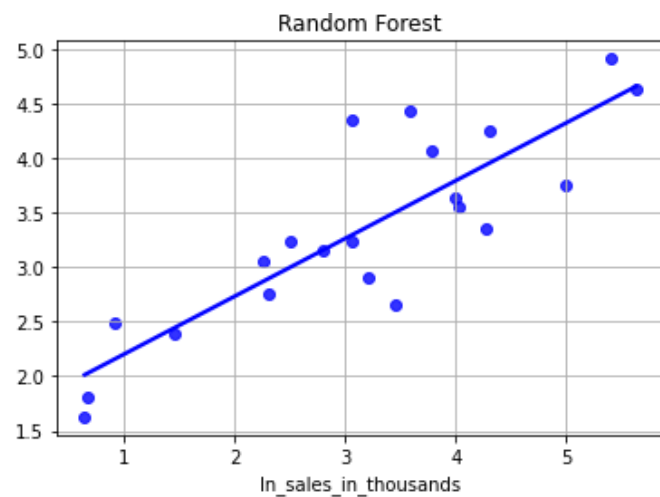
Support Vector Regression:

R2 Score: 0.491036

	Actual Value	Predicted Value
37	4.279246	4.155770
97	0.926637	2.319897
56	5.626711	4.485645
55	5.401100	5.193415
33	3.997834	3.466799
84	2.512603	3.369235
3	2.260512	3.591530
4	3.063251	3.175322
72	1.466491	2.586930
59	4.306805	4.368403
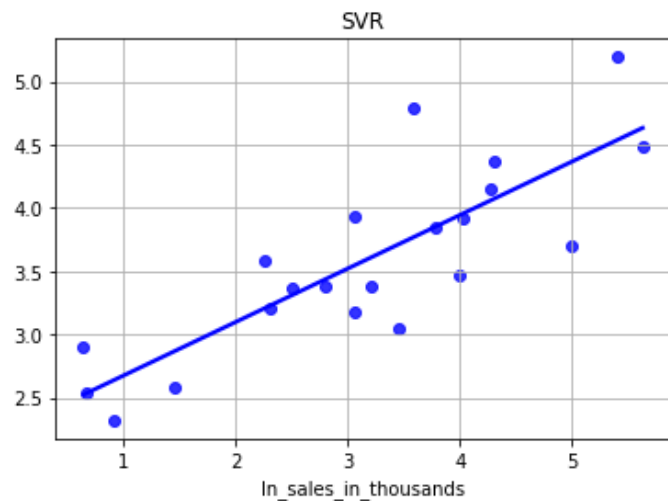
Decision Tree Regression:

R2 Score: 0.499998

	Actual Value	Predicted Value
37	4.279246	4.584217
97	0.926637	1.436137
56	5.626711	5.232836
55	5.401100	4.818046
33	3.997834	3.212093
84	2.512603	3.212093
3	2.260512	3.652729
4	3.063251	3.009869
72	1.466491	3.009869
59	4.306805	5.301737
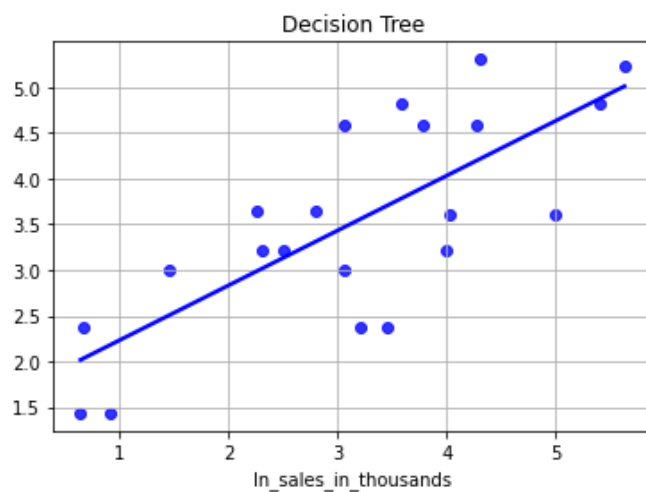
COCLUSION OF THE RESULTS

I studied and implemented Various supervised machine learning algorithms to predict and estimate sales of cars. Data was split into test and train and after necessary implementation steps, I found out that Linear Regression is able to most accurately predict the sales for this particular type of data-set. Any such data set that consists of industry-based most impactful features and a general mix of manufacturers will be accepted by the algorithm and a similar prediction can be replicated.

After implementing and comparing the accuracy of various methods, we come to the following Result..

Model	R2 Score
Linear Regression	0.66306
Adaboost Regression	0.61048
Random Forest	0.65868
KNeighbors	0.55823
Support Vector Regression	0.49103
Decision Tree Regression	0.49999

Linear Regression provides the best accuracy at this point in time in the study. Random Forest and Adaboost Regression also contribute to the ongoing study path.

	test values	Linear Regression	Adaboost	Random Forest	KNeighbors	SVR	Decision Tree
37	4.279246	3.697881	3.240416	3.359184	3.654227	4.155770	4.584217
97	0.926637	2.481541	2.782364	2.493229	3.500414	2.319897	1.436137
56	5.626711	6.523986	4.741314	4.634086	3.657452	4.485645	5.232836
55	5.401100	5.412050	4.860065	4.912153	3.687144	5.193415	4.818046
33	3.997834	3.362431	3.659949	3.639119	3.634456	3.466799	3.212093
84	2.512603	2.840881	3.054022	3.234227	3.515711	3.369235	3.212093
3	2.260512	3.481335	3.237238	3.060825	3.586826	3.591530	3.652729
4	3.063251	2.512318	3.160330	3.233639	3.507082	3.175322	3.009869
72	1.466491	2.667088	2.196221	2.392597	3.457464	2.586930	3.009869
59	4.306805	4.311010	3.488545	4.248885	3.658822	4.368403	5.301737

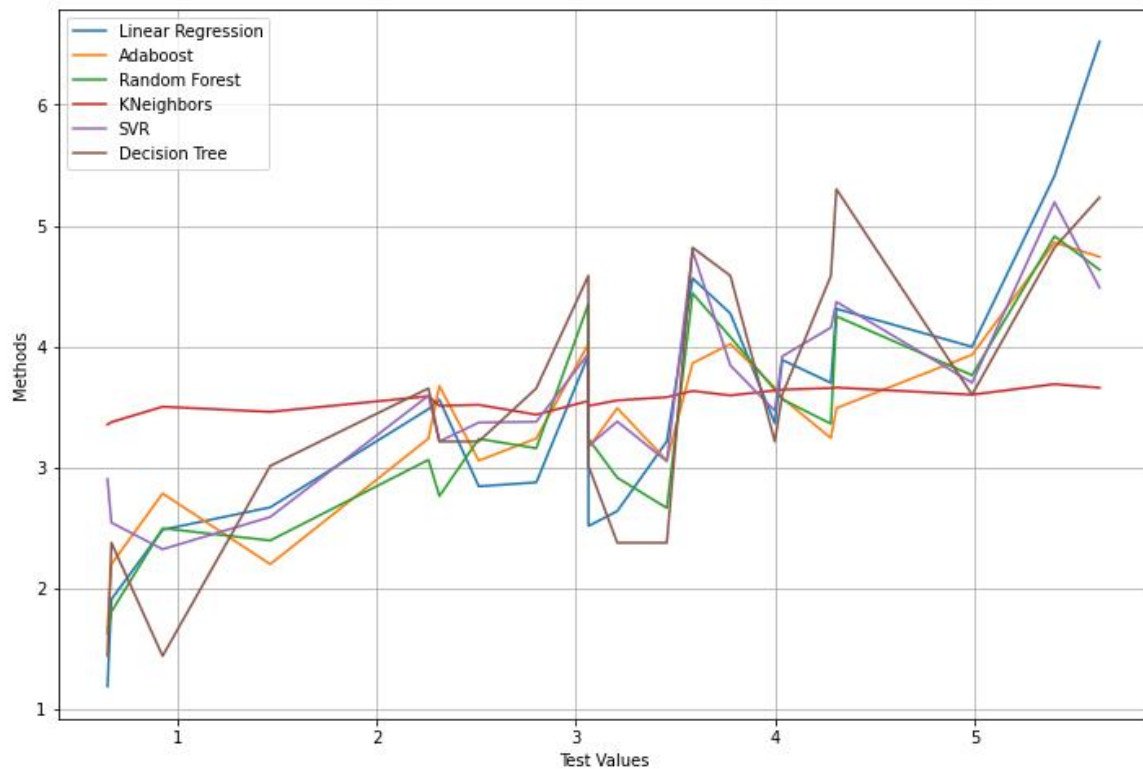Image Shows test values and the predicted values of various methods .

Fig. shows all methods Predicted values vs the actual test values lineplot.

Future Scope:

The algorithms can be combined and implemented to improve the accuracy of prediction as well. By giving more time and resources to this study, an approximate measurement can be made to list down top key parameters that affect the sales of cars for this particular type of data-set. Such a study flow will prove beneficial to car manufacturers very effectively.