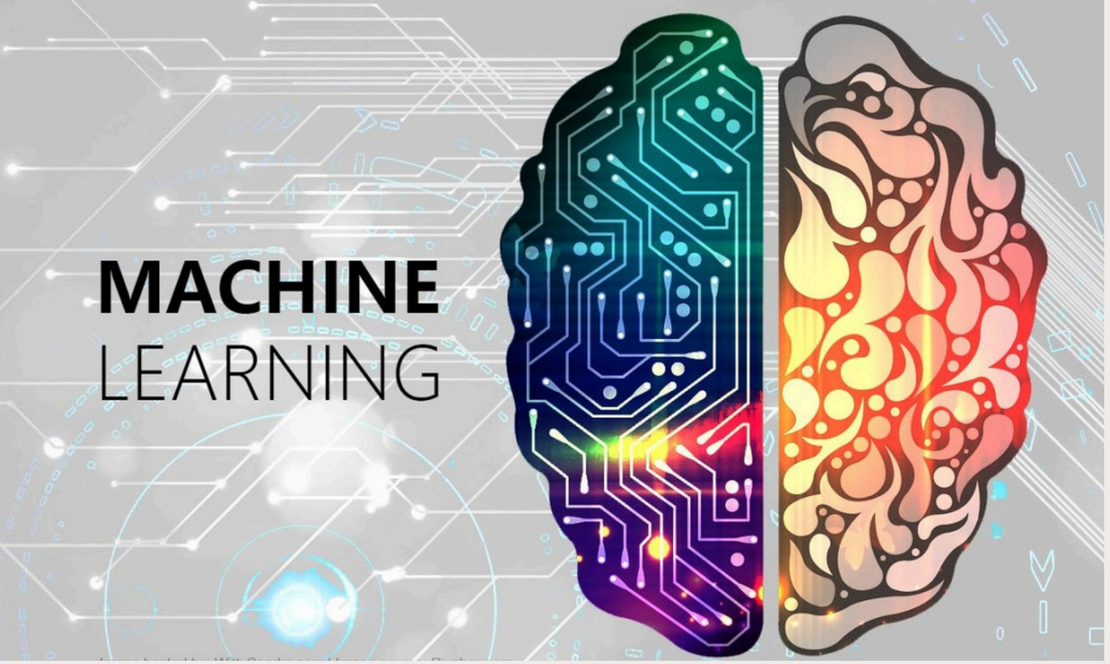


ADHWAITHA .A

DA-DS

MORNING BATCH

Machine Learning

Supervised

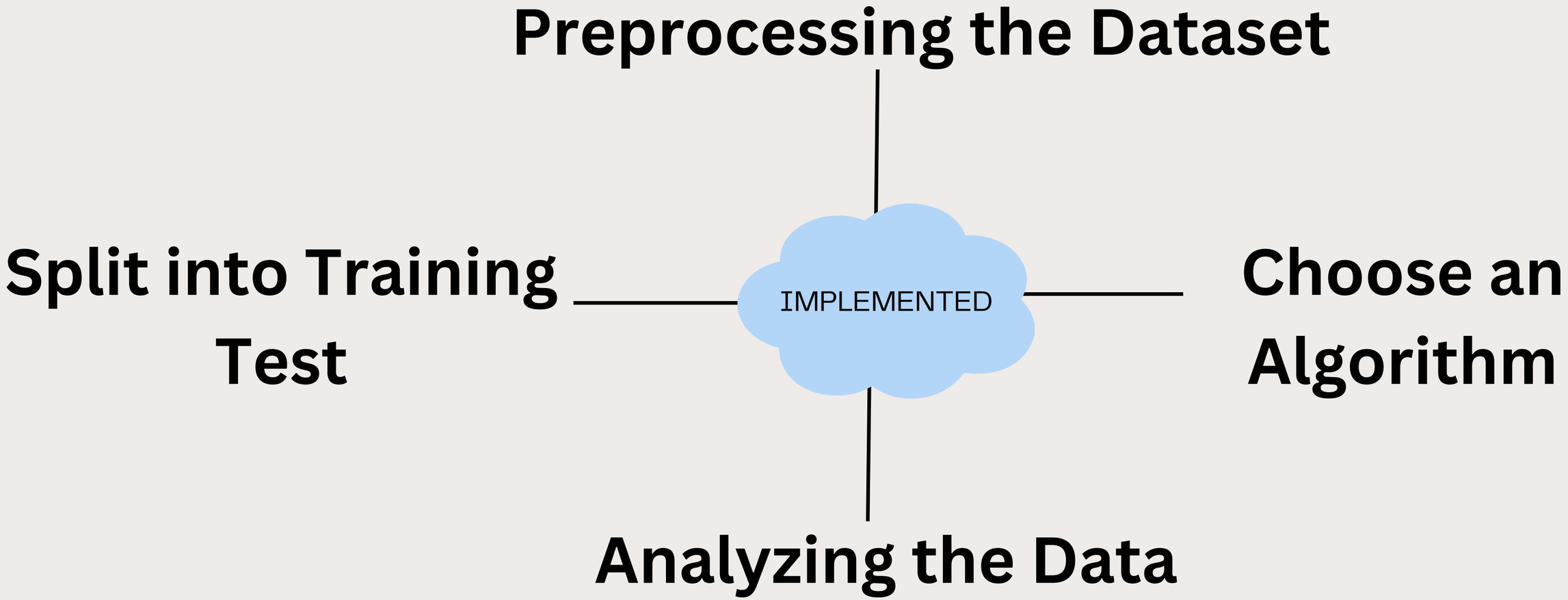
Unsupervised

Semisupervised

DATA SETS

CAR

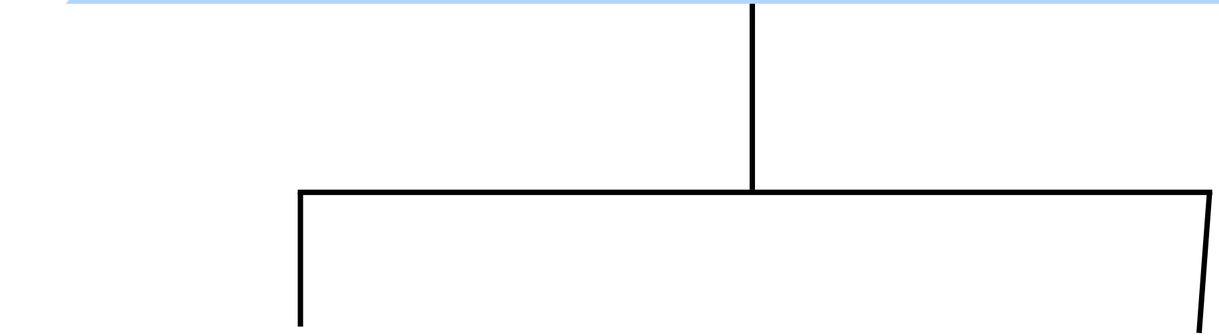
CROP


Preprocessing the Dataset
Split into Training Test
IMPLEMENTED
Choose an Algorithm
Analyzing the Data

Algorithms

1.Logistic Regression- Binary classification that predict the probability of an outcome
2. Linear Regression-Predicting a continuous value based on linear relation with factors.
3. Decision Tree-makes decisions through a series of questions about features.
4.Random Forest-Combines many decision trees for accuracy.
5. Navie bases-Classifies using probabilities, assumes independence
6. Support vector Machine-Finds the best line to separate groups
7. KNN-Classifies based on the closest items.

Supervised learning



Regression

- Linear
- Logistic
- Decision Tree
- Random Forest
- Lasso

Classification

- KNN
- Naive Bayes
- Decision Tree
- Random Forest
- Logistic

Car Dataset

jupyter car_task (1) Last Checkpoint: yesterday

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
```

[104]:
```
df=pd.read_csv("C:/Users/user/Downloads/Car details v3.csv")
```

[4]:
```
df
```

[4]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda City	2006	158000	140000	Petrol	Individual	Manual	Third	17.7	1497	78 bhp	12.7@ 2,700(kgm@	5.0

Data Cleaning

jupyter car_task (1) Last Checkpoint: yesterday

File Edit View Run Kernel Settings Help

Code JupyterLab

8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57	1396 CC	70 bhp

7906 rows × 13 columns

[15]:
```
df.loc[:, 'torque'] = pd.to_numeric(df['torque'].astype(str).str.split().str[0].str.replace('Nm@', ''), errors='coerce')
```

[16]:
```
df.torque.info()
```

```
<class 'pandas.core.series.Series'>
Index: 7906 entries, 0 to 8127
Series name: torque
Non-Null Count  Dtype
--------------  -----
7151 non-null   object
dtypes: object(1)
memory usage: 123.5+ KB
```

[17]:
```
df.loc[:, 'max_power'] = pd.to_numeric(df['max_power'].astype(str).str.replace('bhp', '').str.strip(), errors='coerce')
```

[18]:
```
df.max_power.info()
```

```
<class 'pandas.core.series.Series'>
Index: 7906 entries, 0 to 8127
```

localhost:8888/notebooks/Downloads/car_task%20(1).ipynb

jupyter car_task (1) Last Checkpoint: yesterday

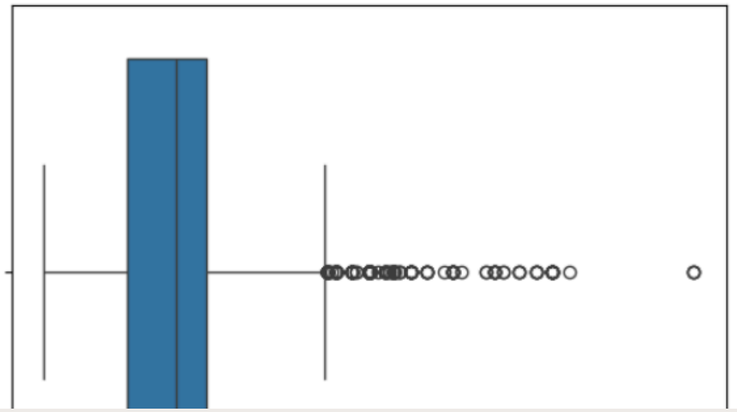
File Edit View Run Kernel Settings Help

Code

```
max_power        0
torque         755
seats            0
dtype: int64
```

[23]:
```
sns.boxplot(x=df['torque'])
```

[23]: <Axes: xlabel='torque'>



Jupyter car_task (1) Last Checkpoint: yesterday

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[29]:
```
df['mileage'].sort_values(ascending=True)
```

[29]:
```
4527    0.0
2725    0.0
6824    0.0
785     0.0
6629    0.0
       ...
7308    NaN
7543    NaN
7642    NaN
7733    NaN
7913    NaN
Name: mileage, Length: 7906, dtype: object
```

[30]:
```
a=df.mileage.median()
a
```

[30]:
```
19.3
```

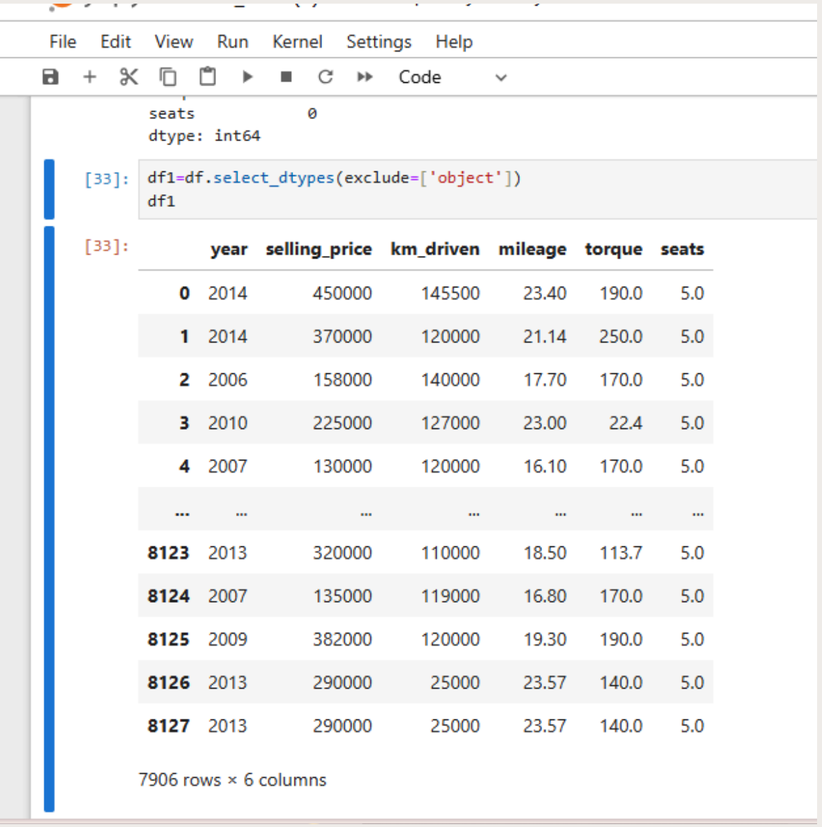
[31]:
```
df.mileage.fillna(a,inplace=True)
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_38576\1349450490.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.mileage.fillna(a,inplace=True)
```

[32]:
```
df.isnull().sum()
```

[32]:
```
name             0
year             0
selling_price    0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
mileage          0
engine           0
max_power        0
```

Outlier Detection and Removal



File Edit View Run Kernel Settings Help

Code

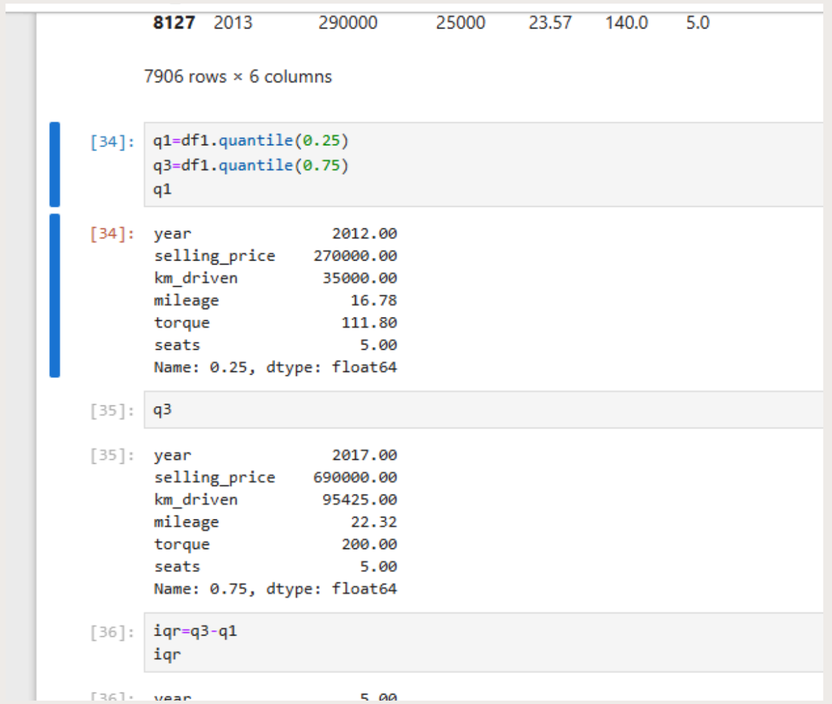
```
seats           0
dtype: int64
```

[33]:
```
df1=df.select_dtypes(exclude=['object'])
df1
```

[33]:

	year	selling_price	km_driven	mileage	torque	seats
0	2014	450000	145500	23.40	190.0	5.0
1	2014	370000	120000	21.14	250.0	5.0
2	2006	158000	140000	17.70	170.0	5.0
3	2010	225000	127000	23.00	22.4	5.0
4	2007	130000	120000	16.10	170.0	5.0
...	...	...	...	...	...	...
8123	2013	320000	110000	18.50	113.7	5.0
8124	2007	135000	119000	16.80	170.0	5.0
8125	2009	382000	120000	19.30	190.0	5.0
8126	2013	290000	25000	23.57	140.0	5.0
8127	2013	290000	25000	23.57	140.0	5.0

7906 rows × 6 columns



8127	2013	290000	25000	23.57	140.0	5.0

7906 rows × 6 columns

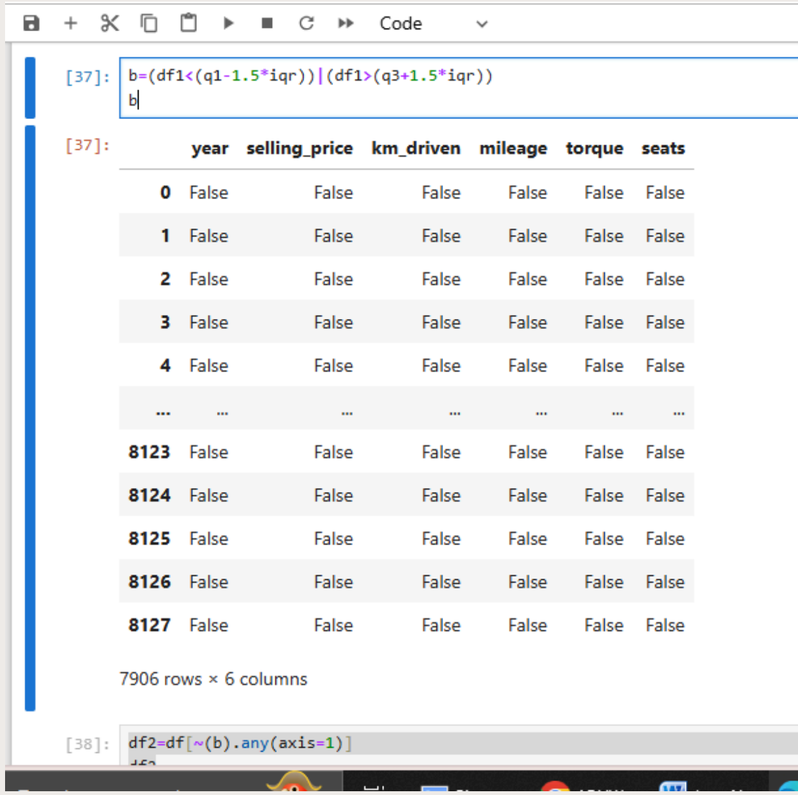
[34]:
```
q1=df1.quantile(0.25)
q3=df1.quantile(0.75)
q1
```

[34]:
```
year              2012.00
selling_price    270000.00
km_driven         35000.00
mileage              16.78
torque              111.80
seats                 5.00
Name: 0.25, dtype: float64
```

[35]:
```
q3
```

[35]:
```
year              2017.00
selling_price    690000.00
km_driven         95425.00
mileage              22.32
torque              200.00
seats                 5.00
Name: 0.75, dtype: float64
```

[36]:
```
iqr=q3-q1
iqr
```



Code

[37]:
```
b=(df1<(q1-1.5*iqr))|(df1>(q3+1.5*iqr))
b
```

[37]:

	year	selling_price	km_driven	mileage	torque	seats
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
8123	False	False	False	False	False	False
8124	False	False	False	False	False	False
8125	False	False	False	False	False	False
8126	False	False	False	False	False	False
8127	False	False	False	False	False	False

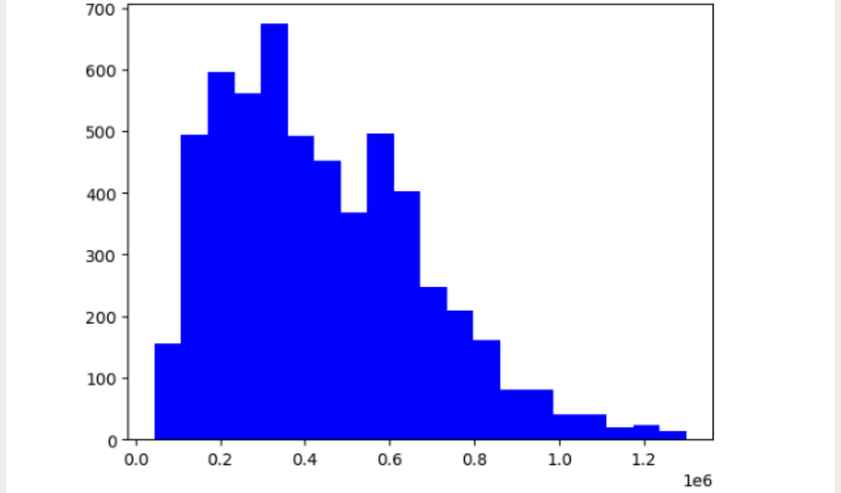
7906 rows × 6 columns

[38]:
```
df2=df[~(b).any(axis=1)]
```

(df1<(q1-1.5*iqr))|(df1>(q3+1.5*iqr)) : Values lower than or higher than are considered outliers.

Data Visualization

```
[39]: plt.hist(df2['selling_price'],bins=20,color='blue')

[39]: (array([156., 495., 596., 561., 674., 492., 453., 368., 496., 403., 248.,
        209., 161.,  81.,  80.,  40.,  40.,  20.,  23.,  14.]),
 array([  45000.,  107750.,  170500.,  233250.,  296000.,  358750.,
         421500.,  484250.,  547000.,  609750.,  672500.,  735250.,
         798000.,  860750.,  923500.,  986250., 1049000., 1111750.,
        1174500., 1237250., 1300000.]),
 <BarContainer object of 20 artists>)
```



plt.hist(): Displays the distribution of selling_price.

sns.pairplot(): Visualizes the relationships between numerical columns.

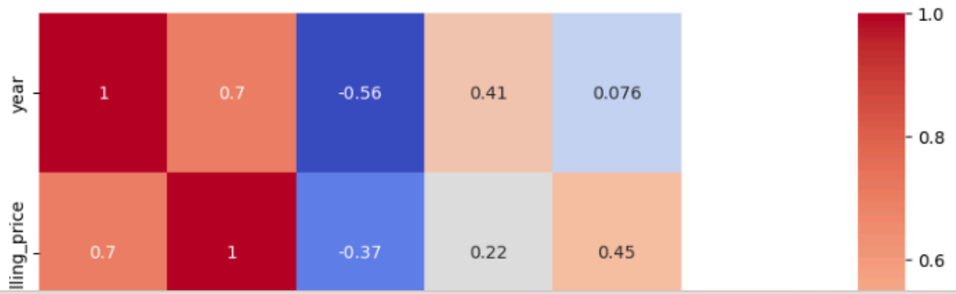
[47]:
```
df1.corr()
```

[47]:

	year	selling_price	km_driven	mileage	torque	seats
year	1.000000	0.697896	-0.557120	0.406794	0.075960	NaN
selling_price	0.697896	1.000000	-0.371182	0.225000	0.447797	NaN
km_driven	-0.557120	-0.371182	1.000000	-0.071230	0.244400	NaN
mileage	0.406794	0.225000	-0.071230	1.000000	0.169434	NaN
torque	0.075960	0.447797	0.244400	0.169434	1.000000	NaN
seats	NaN	NaN	NaN	NaN	NaN	NaN

[48]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df1.corr(),annot=True,cmap='coolwarm')
```

[48]:
```
<Axes: >
```



sns.heatmap(): correlations between numerical variables

year selling_price km_driven mileage torque

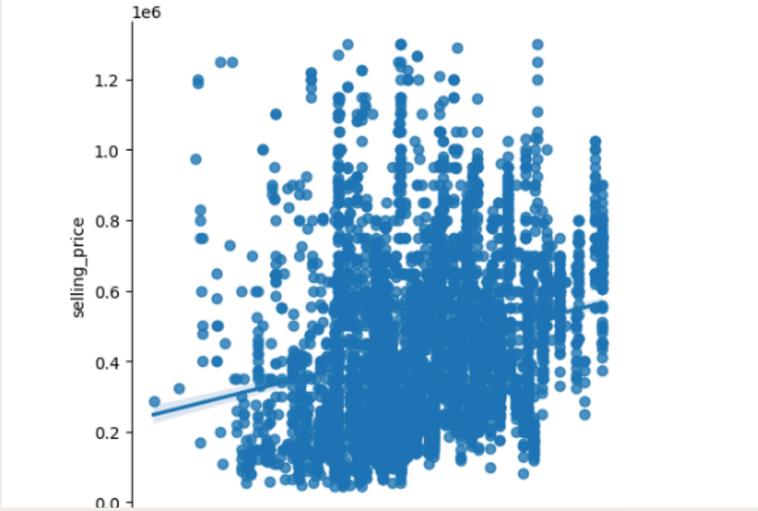
```
[49]: sns.lmplot(x='km_driven',y='year',data=df1)
```

```
[49]: <seaborn.axisgrid.FacetGrid at 0x24dae464810>
```


```
[50]: Index(['year', 'selling_price', 'km_driven', 'mileage', 'torque', 'seats'], dtype=
```

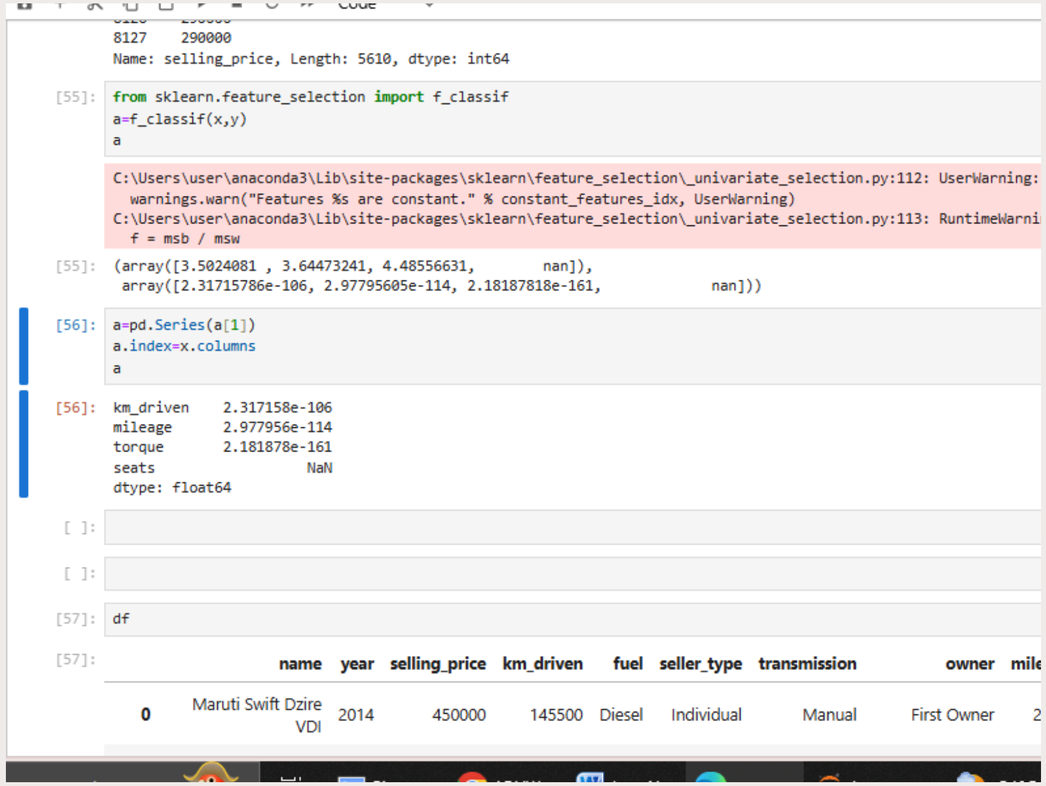
```
[51]: sns.lmplot(x='mileage', y='selling_price', data=df1)
```

```
[51]: <seaborn.axisgrid.FacetGrid at 0x24daf41ec90>
```



lmplot:linear relationship between two variables along with a regression line.

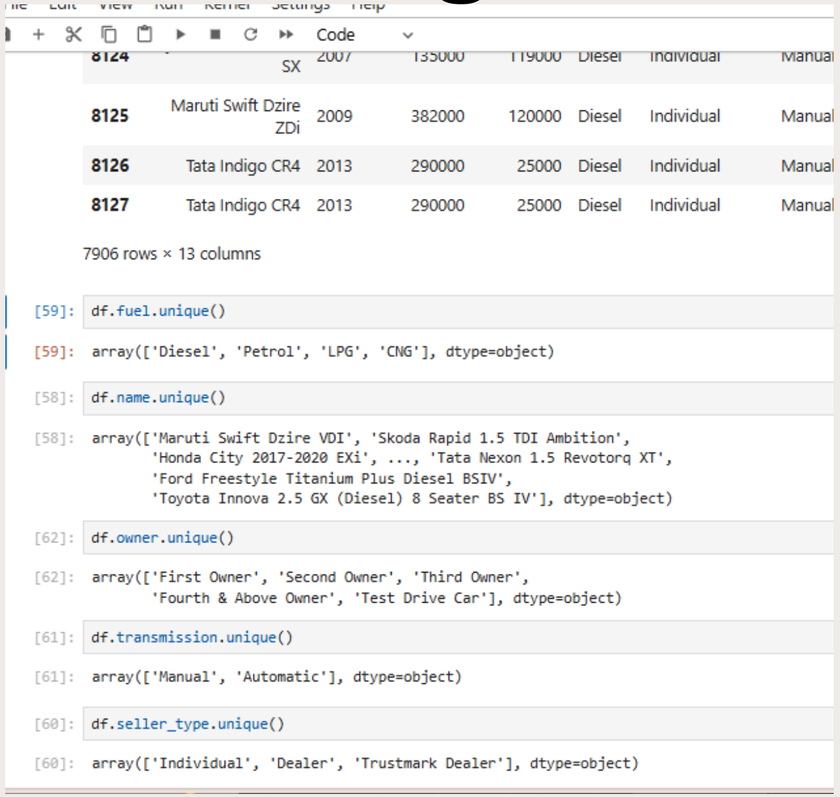
Feature Selection and Scaling



```
8127    290000
Name: selling_price, Length: 5610, dtype: int64

[55]: from sklearn.feature_selection import f_classif
      a=f_classif(x,y)
      a

C:\Users\user\anaconda3\Lib\site-packages\sklearn\feature_selection\_univariate_selection.py:112: UserWarning:
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
C:\Users\user\anaconda3\Lib\site-packages\sklearn\feature_selection\_univariate_selection.py:113: RuntimeWarni
  f = msb / msw

[55]: (array([3.5024081 , 3.64473241, 4.48556631,        nan]),
       array([2.31715786e-106, 2.97795605e-114, 2.18187818e-161,            nan]))

[56]: a=pd.Series(a[1])
      a.index=x.columns
      a

[56]: km_driven    2.317158e-106
      mileage      2.977956e-114
      torque       2.181878e-161
      seats                  NaN
      dtype: float64

[ ]:

[ ]:

[57]: df
```

[57]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mile
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	2



Code

8124	SX	2007	135000	119000	Diesel	Individual	Manual
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual
8126	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual

7906 rows × 13 columns

```
[59]: df.fuel.unique()

[59]: array(['Diesel', 'Petrol', 'LPG', 'CNG'], dtype=object)

[58]: df.name.unique()

[58]: array(['Maruti Swift Dzire VDI', 'Skoda Rapid 1.5 TDI Ambition',
             'Honda City 2017-2020 EXi', ..., 'Tata Nexon 1.5 Revotorq XT',
             'Ford Freestyle Titanium Plus Diesel BSIV',
             'Toyota Innova 2.5 GX (Diesel) 8 Seater BS IV'], dtype=object)

[62]: df.owner.unique()

[62]: array(['First Owner', 'Second Owner', 'Third Owner',
             'Fourth & Above Owner', 'Test Drive Car'], dtype=object)

[61]: df.transmission.unique()

[61]: array(['Manual', 'Automatic'], dtype=object)

[60]: df.seller_type.unique()

[60]: array(['Individual', 'Dealer', 'Trustmark Dealer'], dtype=object)
```

f_classif from sklearn.feature_selecti on : feature selection based on the ANOVA .

- **x: independent variables.**
- **y: dependent variable.**

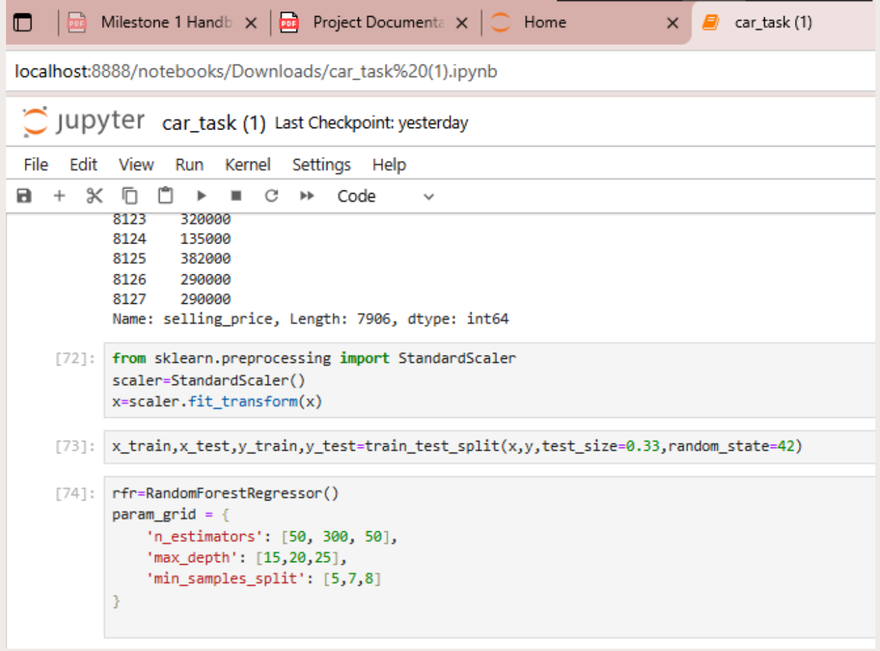
[69]:
```
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

[70]:
```
x=df.drop(columns=['selling_price'])
y=df['selling_price']
x
```

[70]:

	year	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque
0	2014	145500	1	1	1	0	23.40	1248.0	74.0	190.0
1	2014	120000	1	1	1	2	21.14	1498.0	103.52	250.0
2	2006	140000	3	1	1	4	17.70	1497.0	78.0	170.0
3	2010	127000	1	1	1	0	23.00	1396.0	90.0	22.4
4	2007	120000	3	1	1	0	16.10	1298.0	88.2	170.0
...	...	...	...	...	...	...	...	...	...	...
8123	2013	110000	3	1	1	0	18.50	1197.0	82.85	113.7
8124	2007	119000	1	1	1	1	16.80	1493.0	110.0	170.0
8125	2009	120000	1	1	1	0	19.30	1248.0	73.9	190.0
8126	2013	25000	1	1	1	0	23.57	1396.0	70.0	140.0
8127	2013	25000	1	1	1	0	23.57	1396.0	70.0	140.0

7906 rows × 10 columns



StandardScaler: Scales features to have a mean of 0 and standard deviation of 1

param_grid: Specifies a range of hyperparameters

```
[77]: grid_search.best_params_

[77]: {'max_depth': 20, 'min_samples_split': 5, 'n_estimators': 300}

[78]: grid_search.best_score_

[78]: -26737815091.785797

[79]: pr=grid_search.predict(x_test)

[80]: pr

[80]: array([539573.11422664, 531749.96695635, 167581.17739791, ...,
             593376.61110053, 394402.00414345, 101755.20071576])

[81]: mean_squared_error(y_test,pr)

[81]: 18751994085.315895

[82]: r2_score(y_test,pr)

[82]: 0.9719219152704778

[83]: rfr.fit(x_train,y_train)

[83]: ▾ RandomForestRegressor
      RandomForestRegressor()

[84]: pr=rfr.predict(x_test)
```

- grid_search.best_params_: Shows the optimal hyperparameters found by GridSearchCV.
- grid_search.best_score_: Shows the best score achieved with those hyperparameters.
- mean_squared_error(): Calculates how well the model predicted the test data (lower MSE is better).
- r2_score(): Shows how well the predictions fit the actual data

```
[ ]:
```

```
[88]: from sklearn.tree import DecisionTreeRegressor
```

```
[89]: dt = DecisionTreeRegressor()
```

```
[90]: dt.fit(x_train, y_train)
```

```
[90]: DecisionTreeRegressor
      DecisionTreeRegressor()
```

```
[91]: pr=dt.predict(x_test)
```

```
[92]: pr
```

```
[92]: array([650000., 510000., 168000., ..., 650000., 350000., 140000.])
```

```
[93]: mean_squared_error(y_test,pr)
```

```
[93]: 25662084253.10888
```

```
[94]: r2_score(y_test,pr)
```

```
[94]: 0.9615751704743143
```

KN Regressor

Project Documenta | Home | car_task (1)

otebooks/Downloads/car_task%20(1).ipynb#KN-Regressor

pyter car_task (1) Last Checkpoint: yesterday

dit View Run Kernel Settings Help

Code

KNN Regressor

```
]: from sklearn.neighbors import KNeighborsRegressor
```

```
]: knn=KNeighborsRegressor()
```

```
]: param={'n_neighbors' : [3,5,7,9],
          'weights' : ['uniform','distance'],
          'algorithm' : ['auto','ball_tree']}
```

```
]: knn1=GridSearchCV(knn,param,cv=5,scoring='neg_mean_squared_error')
   knn1.fit(x_train,y_train)
```

```
]: GridSearchCV
   estimator: KNeighborsRegressor
      KNeighborsRegressor
```

```
]: knn1.best_params_
```

```
]: {'algorithm': 'ball_tree', 'n_neighbors': 3, 'weights': 'distance'}
```

```
]: knn1.best_score_
```

```
]: -41954849166.251816
```

```
]: pr=knn1.predict(x_test)
   mean_squared_error(y_test,pr)
```

```
]: 31233827831.19562
```

```
]: r2_score(y_test,pr)
```

```
]: 0.9532323836984163
```

Conclusion

The best performance was of the Random Forest Regressor. It reported the lowest MSE at 18.2 billion and the highest R^2 score at 0.9727, which means it gives the most accurate prediction, up to the explanation of variance in selling_price.

CROP DATASET

Crop Dataset - Random Forest Classification

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import accuracy_score, classification_report
```

```
df=pd.read_csv("C:/Users/user/Downloads/Crop_recommendation.csv")
```

```
df
```

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice
...	...	...	...	...	...	...	...	...
2195	107	34	32	26.774637	66.413269	6.780064	177.774507	coffee

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2200 entries, 0 to 2199
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   N            2200 non-null   int64
 1   P            2200 non-null   int64
 2   K            2200 non-null   int64
 3   temperature  2200 non-null   float64
 4   humidity     2200 non-null   float64
 5   ph           2200 non-null   float64
 6   rainfall     2200 non-null   float64
 7   label        2200 non-null   object
dtypes: float64(4), int64(3), object(1)
memory usage: 137.6+ KB
```

```
df.isnull().sum()
```

```
N              0
P              0
K              0
temperature    0
humidity       0
ph             0
rainfall       0
label          0
dtype: int64
```

```
df.describe()
```

	N	P	K	temperature	humidity
count	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000
mean	50.551818	53.362727	48.149091	25.616244	71.481779

```
df.describe()
```

	N	P	K	temperature	humidity	ph	rainfall
count	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000
mean	50.551818	53.362727	48.149091	25.616244	71.481779	6.469480	103.463655
std	36.917334	32.985883	50.647931	5.063749	22.263812	0.773938	54.958389
min	0.000000	5.000000	5.000000	8.825675	14.258040	3.504752	20.211267
25%	21.000000	28.000000	20.000000	22.769375	60.261953	5.971693	64.551686
50%	37.000000	51.000000	32.000000	25.598693	80.473146	6.425045	94.867624
75%	84.250000	68.000000	49.000000	28.561654	89.948771	6.923643	124.267508
max	140.000000	145.000000	205.000000	43.675493	99.981876	9.935091	298.560117

```
df = df.dropna()
```

```
df
```

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice

Code

```
0
```

```
from sklearn.preprocessing import LabelEncoder
```

```
label_encoder = LabelEncoder()
```

```
df.columns
```

```
Index(['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall', 'label'], dtype='object')
```

```
df['label'] = label_encoder.fit_transform(df['label'])
```

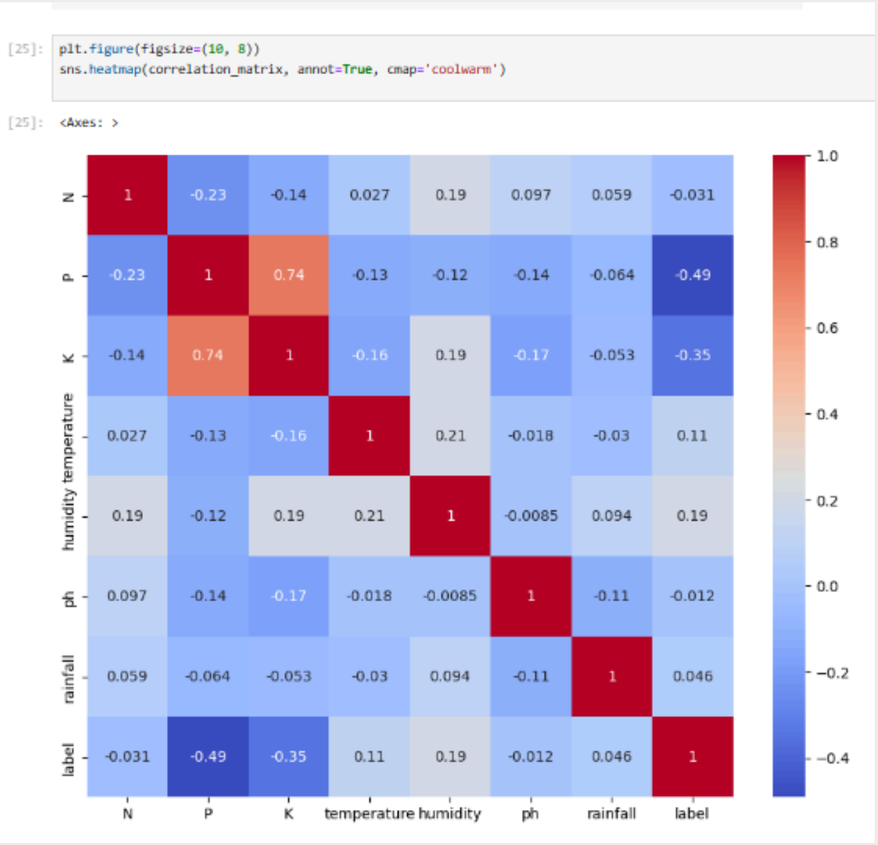
```
df['label']
```

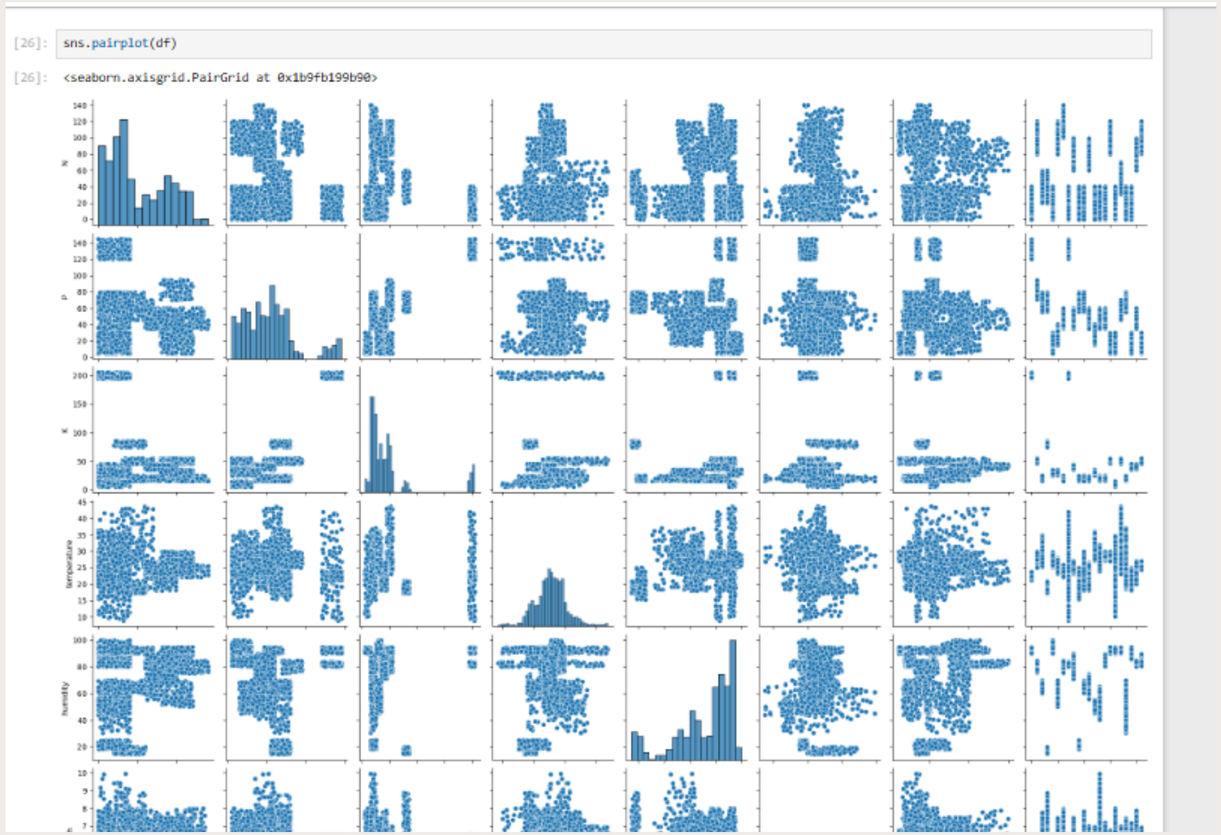
```
0       20
1       20
2       20
3       20
4       20
        ..
2195     5
2196     5
2197     5
2198     5
2199     5
Name: label, Length: 2200, dtype: int32
```

```
df.isnull().sum()
```

```
N              0
P              0
K              0
temperature    0
```

Visualize Data





Heatmap : correlation matrix of a dataset

Pairplot : visualize the relationships between multiple pairs of variables.

Split Data

X : features

Y : Target variable

```
X = df[['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall']]
```

```
X
```

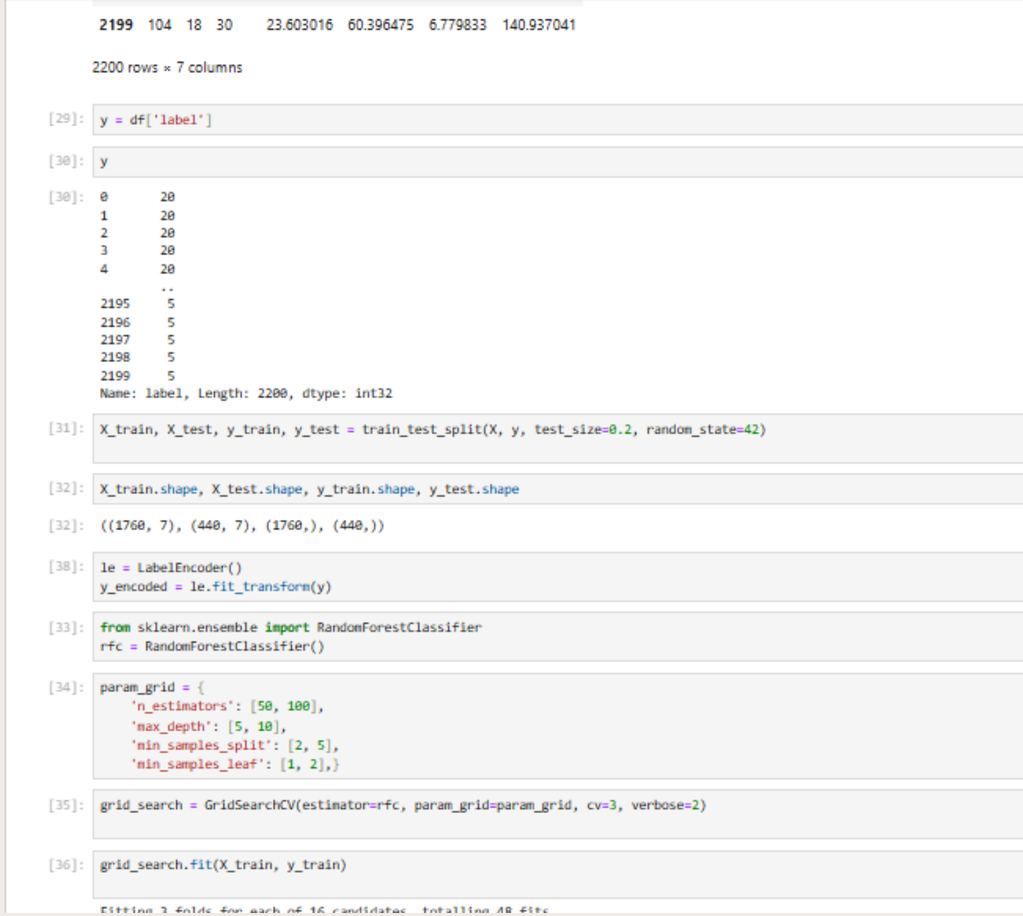
	N	P	K	temperature	humidity	ph	rainfall
0	90	42	43	20.879744	82.002744	6.502985	202.935536
1	85	58	41	21.770462	80.319644	7.038096	226.655537
2	60	55	44	23.004459	82.320763	7.840207	263.964248
3	74	35	40	26.491096	80.158363	6.980401	242.864034
4	78	42	42	20.130175	81.604873	7.628473	262.717340
...	...	...	...	...	...	...	...
2195	107	34	32	26.774637	66.413269	6.780064	177.774507
2196	99	15	27	27.417112	56.636362	6.086922	127.924610
2197	118	33	30	24.131797	67.225123	6.362608	173.322839
2198	117	32	34	26.272418	52.127394	6.758793	127.175293
2199	104	18	30	23.603016	60.396475	6.779833	140.937041

2200 rows × 7 columns

```
y = df['label']
```

```
y
```

```
0       20
1       20
2       20
3       20
4       20
        ..
2195     5
2196     5
2197     5
2198     5
2199     5
```



```
y = df['label']
```

```
y
```

```
0       20
1       20
2       20
3       20
4       20
        ..
2195     5
2196     5
2197     5
2198     5
2199     5
Name: label, Length: 2200, dtype: int32
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((1760, 7), (440, 7), (1760,), (440,))
```

```
le = LabelEncoder()
y_encoded = le.fit_transform(y)
```

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
```

```
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [5, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],}
```

```
grid_search = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=3, verbose=2)
```

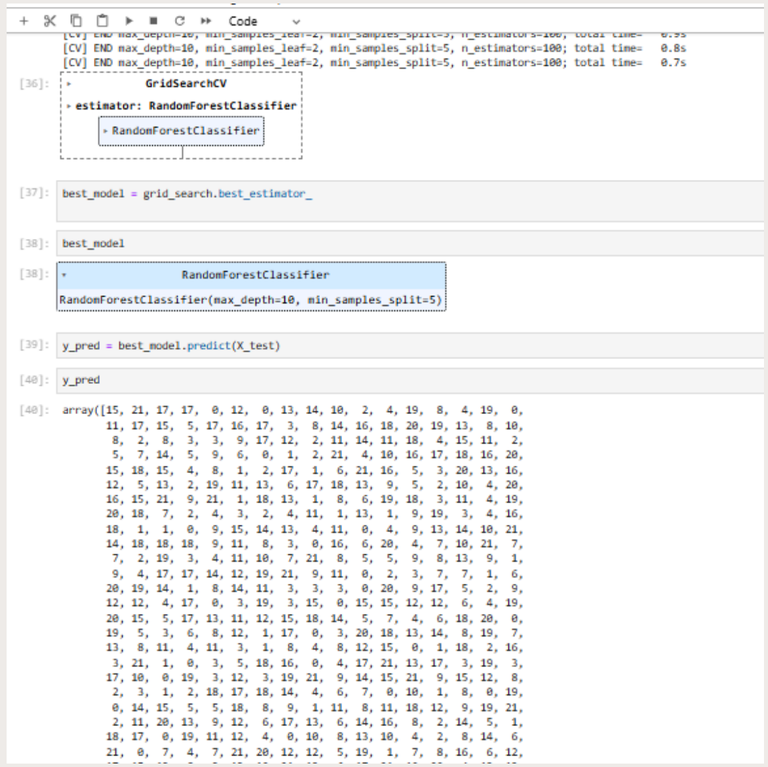
```
grid_search.fit(X_train, y_train)
```

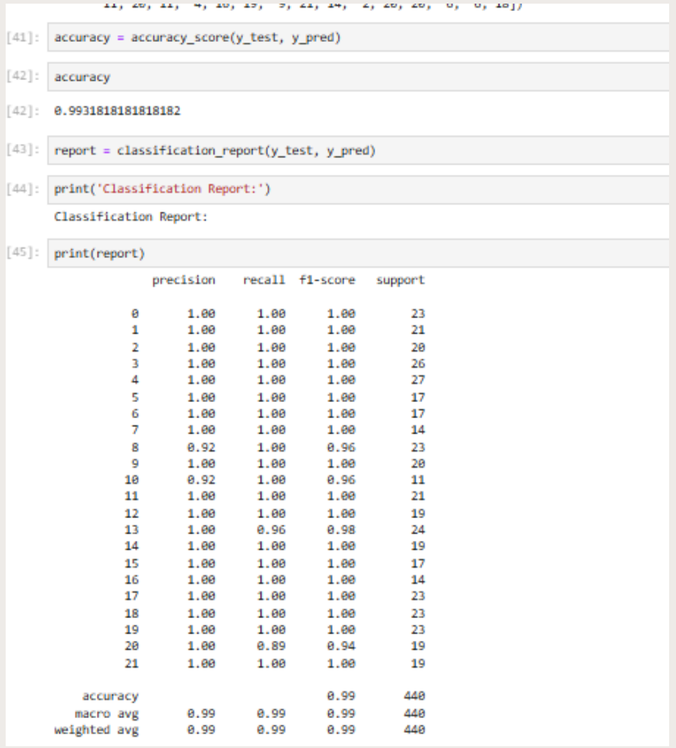
```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
```

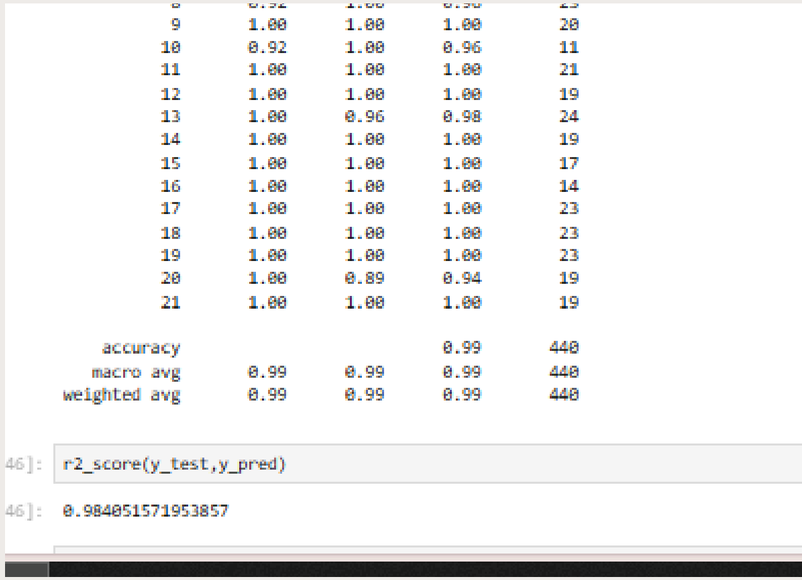
n_estimators :higher number of trees can increase accuracy

max_depth: smaller depth prevents overfitting , larger depth allows to capture more complex patterns but may lead to overfitting.

min_samples_split: Increasing the value can prevent from overfitting.







accuracy : 0.9931818181818182

r2_score : 0.984051571953857

K-Nearest Neighbors (KNN) with GridSearchCV

KNN Classification

```
[52]: from sklearn.neighbors import KNeighborsClassifier

[53]: knn = KNeighborsClassifier()

[54]: param_grid = {
          'n_neighbors': [3, 5, 7, 10],
          'weights': ['uniform', 'distance'],}

[55]: grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=2, verbose=2)

[56]: grid_search.fit(X_train, y_train)

Fitting 2 folds for each of 8 candidates, totalling 16 fits
[CV] END .....n_neighbors=3, weights=uniform; total time=   0.1s
[CV] END .....n_neighbors=3, weights=uniform; total time=   0.1s
[CV] END .....n_neighbors=3, weights=distance; total time=   0.0s
[CV] END .....n_neighbors=3, weights=distance; total time=   0.0s
[CV] END .....n_neighbors=5, weights=uniform; total time=   0.0s
[CV] END .....n_neighbors=5, weights=uniform; total time=   0.1s
[CV] END .....n_neighbors=5, weights=distance; total time=   0.0s
[CV] END .....n_neighbors=5, weights=distance; total time=   0.0s
[CV] END .....n_neighbors=7, weights=uniform; total time=   0.0s
[CV] END .....n_neighbors=7, weights=uniform; total time=   0.0s
[CV] END .....n_neighbors=7, weights=distance; total time=   0.0s
[CV] END .....n_neighbors=7, weights=distance; total time=   0.0s
[CV] END .....n_neighbors=10, weights=uniform; total time=   0.1s
[CV] END .....n_neighbors=10, weights=uniform; total time=   0.1s
[CV] END .....n_neighbors=10, weights=distance; total time=   0.0s
[CV] END .....n_neighbors=10, weights=distance; total time=   0.0s
[56]: GridSearchCV
      estimator: KNeighborsClassifier
          KNeighborsClassifier

[57]: best_est = grid_search.best_estimator_

[58]: best_est
```

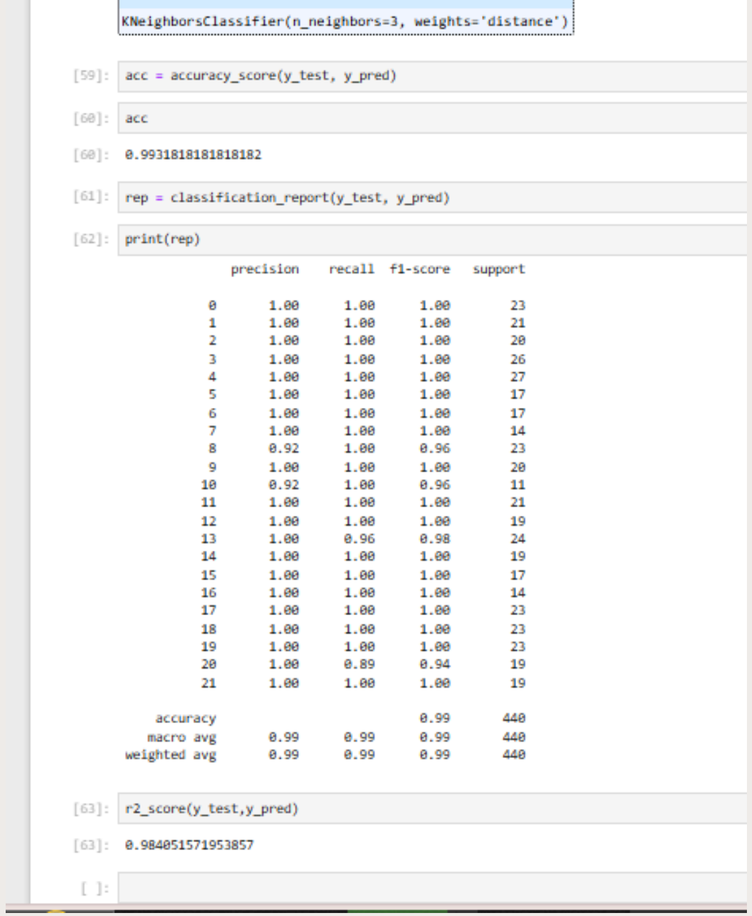
```
[CV] END .....n_neighbors=10, weights=distance; total time=   0.0s
[CV] END .....n_neighbors=10, weights=distance; total time=   0.0s
[56]: GridSearchCV
      estimator: KNeighborsClassifier
          KNeighborsClassifier

[57]: best_est = grid_search.best_estimator_

[58]: best_est

[58]: KNeighborsClassifier
      KNeighborsClassifier(n_neighbors=3, weights='distance')

[59]: acc = accuracy_score(y_test, y_pred)

[60]: acc

[60]: 0.9931818181818182

[61]: rep = classification_report(y_test, y_pred)

[62]: print(rep)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	23
1	1.00	1.00	1.00	21
2	1.00	1.00	1.00	20
3	1.00	1.00	1.00	26
4	1.00	1.00	1.00	27
5	1.00	1.00	1.00	17
6	1.00	1.00	1.00	17
7	1.00	1.00	1.00	14
8	0.92	1.00	0.96	23
9	1.00	1.00	1.00	20
10	0.92	1.00	0.96	11
11	1.00	1.00	1.00	21
12	1.00	1.00	1.00	19

n_neighbors: smaller number makes more sensitive and may occur overfit.

weights:

1. uniform: All neighbors have equal weight in the prediction.
2. distance: Closer neighbors have more weight than those farther away.



```
KNeighborsClassifier(n_neighbors=3, weights='distance')
```

```
[59]: acc = accuracy_score(y_test, y_pred)
```

```
[60]: acc
```

```
[60]: 0.9931818181818182
```

```
[61]: rep = classification_report(y_test, y_pred)
```

```
[62]: print(rep)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        23
           1       1.00      1.00      1.00        21
           2       1.00      1.00      1.00        20
           3       1.00      1.00      1.00        26
           4       1.00      1.00      1.00        27
           5       1.00      1.00      1.00        17
           6       1.00      1.00      1.00        17
           7       1.00      1.00      1.00        14
           8       0.92      1.00      0.96        23
           9       1.00      1.00      1.00        20
          10       0.92      1.00      0.96        11
          11       1.00      1.00      1.00        21
          12       1.00      1.00      1.00        19
          13       1.00      0.96      0.98        24
          14       1.00      1.00      1.00        19
          15       1.00      1.00      1.00        17
          16       1.00      1.00      1.00        14
          17       1.00      1.00      1.00        23
          18       1.00      1.00      1.00        23
          19       1.00      1.00      1.00        23
          20       1.00      0.89      0.94        19
          21       1.00      1.00      1.00        19

    accuracy                           0.99       440
   macro avg       0.99      0.99      0.99       440
weighted avg       0.99      0.99      0.99       440
```

```
[63]: r2_score(y_test,y_pred)
```

```
[63]: 0.984051571953857
```

```
[ ]:
```

accuracy : 0.9931818181818182

r2_score : 0.984051571953857

Conclusion

Both the Random Forest Classifier (RFC) and K-Nearest Neighbors (KNN) classifiers achieved impressive accuracy and R^2 scores, indicating strong predictive performance. Considering the nature and size of the dataset, Random Forest emerges as the superior choice. while KNN offers simplicity and ease of. understanding, the Random Forest Classifier is better equipped to handle the complexities of the dataset

Overview of Both Data set

Comparing the CAR and CROP datasets, the CROP dataset performs better in terms of accuracy. Among the algorithms tested, Random Forest is the best performer for both datasets.


Thank
you