LOAN SANCTION PROJECT

SUBMITTED BY

AVINASH GIRI(0176CD211032)

LNCTE

STEP-1 Import all the necessary libraries such NumPy, pandas, matplotlib, seaborn etc.
Similarly, import the dataset as “loandata.csv”.
As we can see that the head or first 5 entries of the data set.

LOAN SANCTION

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt              ## This all the basic libraries that we will use to create the project
import seaborn as sns
from sklearn import svm
```

[3]:
```
df = pd.read_csv("loandata.csv")             ## Loading the data set
```

[4]:
```
df.head()            ## Prints the first five lines of dataset..
```

[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	Unnamed: 14
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0	NaN
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6	NaN
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	NaN
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	NaN
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2	NaN

STEP-2 Take a look at the dataset as writing the code “df.info()” and it will provide all the details such as how much values are non null values from total values ‘506’ . From the result below we can clearly see that the unnamed attribute seen at index no. 14. This is the noisy data and it will interrupt the whole analysis.

```
[5]: df.info()        ##This info denotes the null values out of 506 values

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   CRIM         486 non-null    float64
 1   ZN           486 non-null    float64
 2   INDUS        475 non-null    float64
 3   CHAS         486 non-null    float64
 4   NOX          506 non-null    float64
 5   RM           506 non-null    float64
 6   AGE          486 non-null    float64
 7   DIS          506 non-null    float64
 8   RAD          506 non-null    int64
 9   TAX          506 non-null    int64
 10  PTRATIO      506 non-null    float64
 11  B            506 non-null    float64
 12  LSTAT        486 non-null    float64
 13  MEDV         506 non-null    float64
 14  Unnamed: 14  0 non-null      float64
dtypes: float64(13), int64(2)
memory usage: 59.4 KB
```

STEP -3 The next step is basically to replace all the “nan” values or null values by ‘0’ by using the data frame command “fillna(0)”

```
[6]: df = df.fillna(0)                    ## filling the null values by 0
     df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   CRIM         506 non-null    float64
 1   ZN           506 non-null    float64
 2   INDUS        506 non-null    float64
 3   CHAS         506 non-null    float64
 4   NOX          506 non-null    float64
 5   RM           506 non-null    float64
 6   AGE          506 non-null    float64
 7   DIS          506 non-null    float64
 8   RAD          506 non-null    int64
 9   TAX          506 non-null    int64
 10  PTRATIO      506 non-null    float64
 11  B            506 non-null    float64
 12  LSTAT        506 non-null    float64
 13  MEDV         506 non-null    float64
 14  Unnamed: 14  506 non-null    float64
dtypes: float64(13), int64(2)
memory usage: 59.4 KB
```

STEP -4 Removing the unnamed entries from the data set.

```
[7]: df.drop("Unnamed: 14", axis=1, inplace=True)          ## Dropping the unnamed column

[8]: df.info()                                             ## After performing the results is our data got cleaned..

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

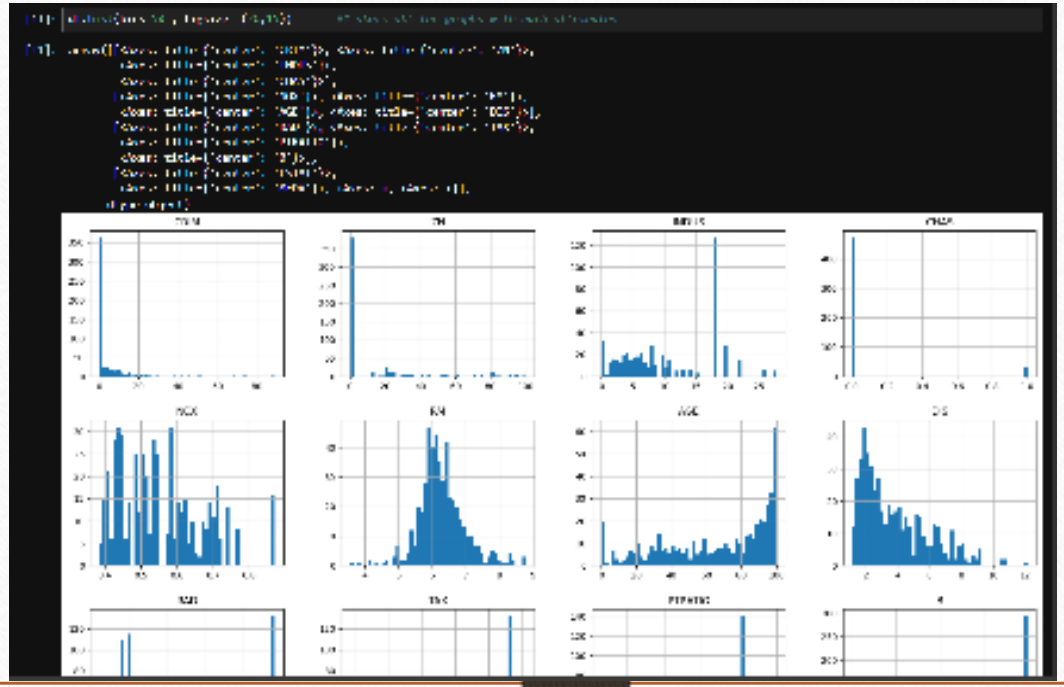
STEP -5 ‘Describe’ command is used to take an overlook of the data like count , mean, standard deviation, min etc.

```
[9]: df.describe()    ## Shows the basic calculation about the data.
```

[9]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	50
mean	3.469112	10.768775	10.495375	0.067194	0.554695	6.284634	65.810277	3.795043	9.549407	408.237154	18.455534	356.674032	12.212846	2
std	8.574755	23.025124	7.181675	0.250605	0.115878	0.702617	30.520606	2.105710	8.707259	168.537116	2.164946	91.294864	7.438324	
min	0.000000	0.000000	0.000000	0.000000	0.385000	3.561000	0.000000	1.129600	1.000000	187.000000	12.600000	0.320000	0.000000	
25%	0.068907	0.000000	4.075000	0.000000	0.449000	5.885500	40.425000	2.100175	4.000000	279.000000	17.400000	375.377500	6.590000	1
50%	0.224635	0.000000	8.560000	0.000000	0.538000	6.208500	74.450000	3.207450	5.000000	330.000000	19.050000	391.440000	10.925000	2
75%	2.808720	0.000000	18.100000	0.000000	0.624000	6.623500	93.575000	5.188425	24.000000	666.000000	20.200000	396.225000	16.570000	2
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	5

STEP – 6 Taking all the graphs related to each attribute and try to read and find out the basic ideas that which attribute affect the most in related dataset or in this ‘price’ .



STEP – 7 Train – Test splitting is the method of dividing the dataset in to two parts :

1. Train part – In this part approx. 70% dataset is taken randomly generated by system which contains all the attributes or we can say that no attribute is being left or neglected.
2. Test part- In this 30% data is left behind for testing the resultant model.

Train- Test splitting

[13]:
```
def split_train_test(data, test_ratio):
    np.random.seed(42)
    shuffled = np.random.permutation(len(data))
    print(shuffled)
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled[:test_set_size]
    train_indices = shuffled[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

[14]:
```
train_set , test_set = split_train_test(df , 0.2)
```

```
[173 274 491  72 452  76 316 140 471 500 218   9 414  78 323 473 124 388
 195 448 271 278  30 501 421 474  79 454 210 497 172 320 375 362 467 153
   2 336 208  73 496 307 204  68  90 390  33  70 470   0  11 281  22 101
 268 485 442 290  84 245  63  55 229  18 351 209 395  82  39 456  46 481
 444 355  77 398 104 203 381 489  69 408 255 392 312 234 460 324  93 137
 176 417 131 346 365 132 371 412 436 411  86  75 477  15 332 423  19 325
 335  56 437 409 334 181 227 434 180  25 493 238 244 250 418 117  42 322
 347 182 155 280 126 329  31 113 148 432 338  57 194  24  17 298  66 211
 404  94 154 441  23 225 433 447   5 116  45  16 468 360   3 405 185  60
 110 321 265  29 262 478  26   7 492 108  37 157 472 118 114 175 192 272
 144 373 383 356 277 220 450 141 369  67 361 168 499 394 400 193 249 109
 420 145  92 152 222 304  83 248 165 163 199 231  74 311 455 253 119 284
 302 483 357 403 228 261 237 386 476  36 196 139 368 247 287 378  59 111
  89 266   6 364 503 341 158 150 177 397 184 318  10 384 103  81  38 317
 167 475 299 296 198 377 146 396 147 428 289 123 490  96 143 239 275  97
 353 122 183 202 246 484 301 354 410 399 286 125 305 223 422 219 129 424
 291 331 380 480 358 297 294 370 438 112 179 310 342 333 487 457 233 314
```

STEP – 9 Setting – up the correlation between the attributes.

STEP – 9 Sometime we can make new attributes by combining previous attributes like “TAXRM” is the new attribute as we can see that and now we check how much it affects the other trends or how much effectively it is.

Attribute Combination

[23]:
```
df["TAXRM"] = df['TAX']/df['RM']
df["TAXRM"]
```

[23]:
```
0      45.019011
1      37.688834
2      33.681280
3      31.723350
4      31.061984
          ...
501    41.407553
502    44.607843
503    39.134174
504    40.182514
505    45.273632
Name: TAXRM, Length: 506, dtype: float64
```

[24]:
```
df.head()
```

[24]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	TAXRM
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0	45.019011
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6	37.688834
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	33.681280
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	31.723350
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	0.00	36.2	31.061984

STEP – 10 Now creating the pipeline using ‘Scikit-learn’ and ‘import standard scalar’ and transform the data in 2 -D array.

Scikit- learn Design and creating a pipeline

[34]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

my_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),

    ('std_scaler', StandardScaler()),
])
```

[35]:
```
df_num_tr = my_pipeline.fit_transform(df)
```

[36]:
```
df_num_tr.shape
```

[36]:
```
(404, 13)
```

```
Models outputs

Decision Tree:
Mean : 5.587444714263071
Standard deviation : 1.396582494024122

Linear Regression :
Mean : 5.190773163830108
Standard deviation : 1.2850828626627553

Random forest Regressor :
Mean : 3.937540396212799
Standard deviation : 1.201510661152486
```

STEP – 11 In our data set which model is getting fit or providing least error will be the best model for our project. We tried

Linear Regression model , Decision Tree Regressor and Random Forest Regressor and we saw that Random Forest Regressor gives us the best result / least error.

Selecting a desired model for problem

[38]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
# model = LinearRegression()
# model = DecisionTreeRegressor()
model.fit(df_num_tr, df_labels)
```

[38]:
```
RandomForestRegressor
RandomForestRegressor()
```

[39]:
```
some_data = df.iloc[:5]
```

[40]:
```
some_labels = df_labels.iloc[:5]
```

[41]:
```
prepared_data = my_pipeline.transform(some_data)
```

[42]:
```
model.predict(prepared_data)
```

[42]:
```
array([ 8.713, 22.637, 27.802, 18.366, 15.838])
```

[43]:
```
list(some_labels)
```

[43]:
```
[8.3, 23.1, 26.6, 17.8, 14.1]
```

STEP – 12 Evaluating the model means getting the exact value of error by calculating the root mean square error (rmse).

Hence, we got the value 2.03.. It means our model is not overfit nor underfit .

Evaluatinf the model

[45]:
```
from sklearn.metrics import mean_squared_error
df_predictions = model.predict(df_num_tr)
mse = mean_squared_error(df_labels, df_predictions)
rmse = np.sqrt(mse)
```

[46]:
```
mse
```

[46]:
```
2.0359486435643546
```

STEP – 13 Cross evaluation technique means comparing the values as we can see that and there is not much difference.

Using better evaluation technique - Cross validation

[48]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, df_num_tr, df_labels, scoring = "neg_mean_squared_error",cv=10)
rmse_scores = np.sqrt(-scores)
```

[49]:
```
rmse_scores
```

[49]:
```
array([5.9108808 , 5.20818014, 4.19137593, 2.9327411 , 5.82041711,
       3.36675433, 2.83797636, 3.50713605, 2.80968069, 2.79026146])
```

[50]:
```
def print_scores(scores):
    print("Scores:", scores)
    print("Mean :", scores.mean())
    print("Standard deviation :", scores.std())
```

[51]:
```
print_scores(rmse_scores)
```

```
Scores: [5.9108808  5.20818014 4.19137593 2.9327411  5.82041711 3.36675433
 2.83797636 3.50713605 2.80968069 2.79026146]
Mean : 3.937540396212799
Standard deviation : 1.201510661152486
```

STEP – 14 Saving the model by import dump and load from “joblib”.

saving the model

[53]:
```
# import sklearn.external.joblib as extjoblib
from  joblib import dump,load
dump(model,'loan_saansaaction.joblib')
```

[53]:
```
['loan_saansaaction.joblib']
```

STEP – 15 As we can see that the data we stored as test data now we use that data to test our model and the resultant as shown below as “final_rmse” score 2.68.. Which decides our model is ready to deploy.

Testing the model with test data

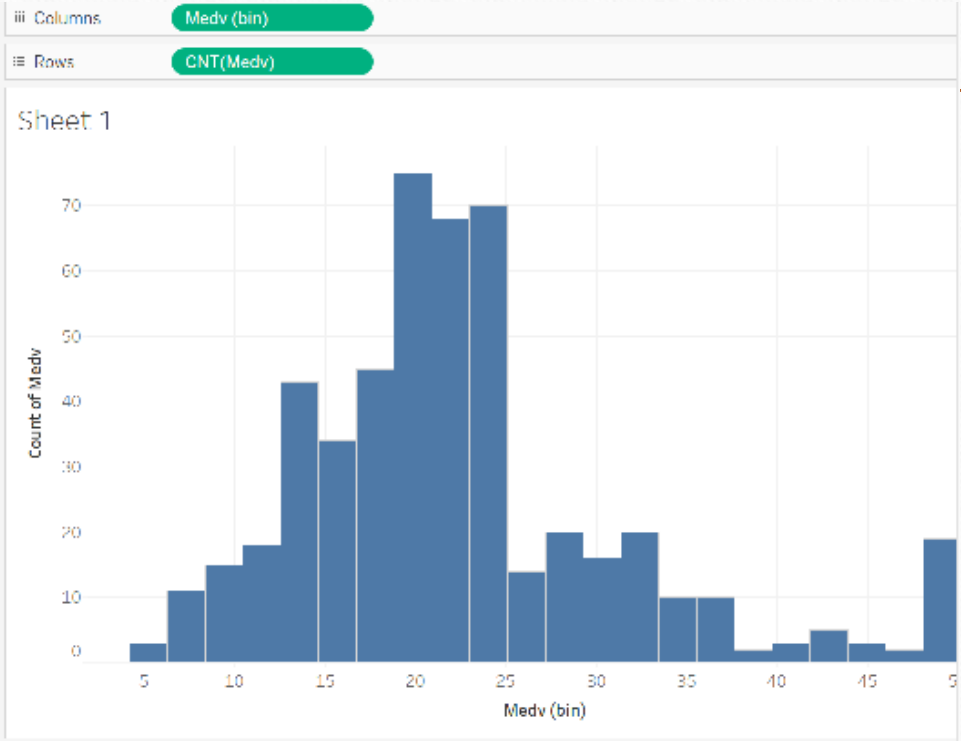
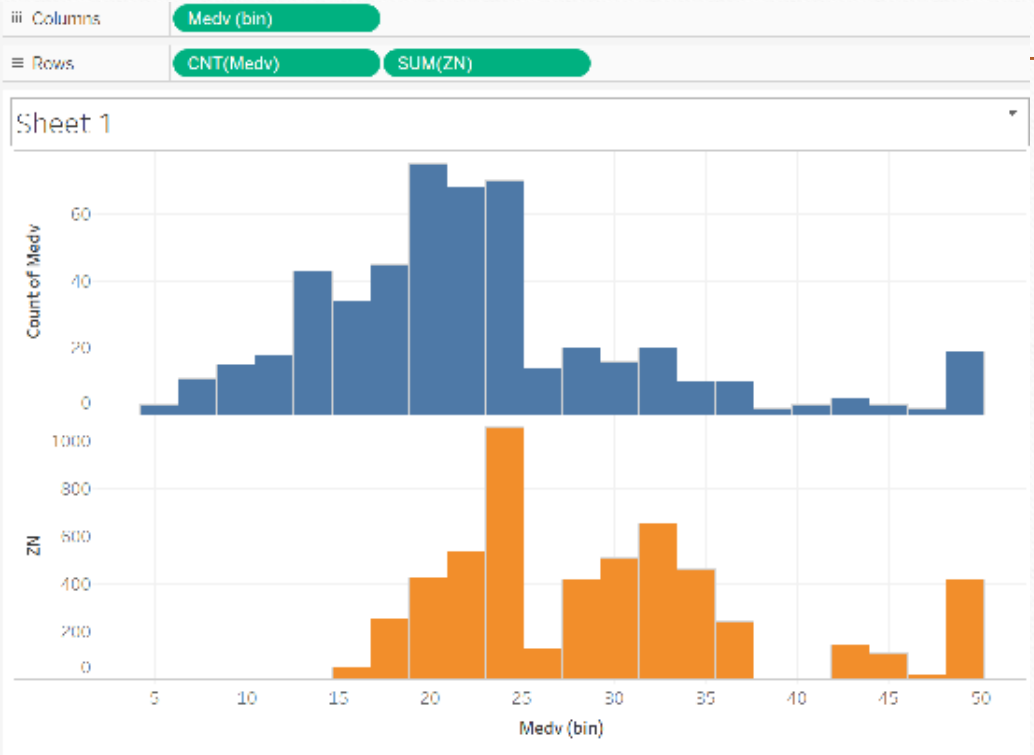
[55]:
```
X_test = strat_test_set.drop("MEDV", axis = 1)
Y_test = strat_test_set["MEDV"].copy()
X_test_prepared = my_pipeline.transform(X_test)
final_predictions = model.predict(X_test_prepared)
final_mse = mean_squared_error(Y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
print(final_predictions , list(Y_test))
```

```
[33.503  9.405 23.758 28.239 18.711 15.227 19.566 15.064 34.553 31.354
 18.66  11.959 19.796 12.767 20.905 11.933 34.669 14.555 20.941 25.831
 19.544 17.766  9.303 24.604 19.893 24.157 17.792 27.168  8.665 33.739
 23.048 21.291 19.912 12.065 20.634 16.662 31.894 25.106 21.52  20.642
 20.694 46.504 20.936 21.441 18.593 23.864 33.723 20.183 20.231 20.227
 21.27  14.868 23.554 20.802 25.927 22.457 47.094 29.514 18.917 20.29
 26.826  8.499 20.018 43.831 14.281 34.56  48.147 17.276 20.9   30.406
 45.75  23.149 26.687 22.34  47.269 14.085 20.469 22.876 23.188 23.763
 22.961 22.054 12.993 24.174 20.831 20.813 12.218 21.719 21.363 23.623
 18.572 26.532 12.595 26.928 16.354 34.704 19.848 31.069 14.014 25.714
 21.842 22.803] [32.7, 13.1, 28.7, 26.7, 14.4, 15.6, 19.4, 12.6, 33.3, 32.4, 23.1, 13.8, 22.6, 11.9, 21.1, 11.3, 35.4, 13.5, 21.0, 27.5, 19.2, 19.5, 10.
2, 29.4, 18.6, 24.8, 15.2, 24.4, 8.8, 30.3, 22.0, 20.6, 23.1, 13.3, 23.8, 13.8, 33.8, 24.7, 19.4, 20.9, 22.5, 48.8, 22.7, 23.4, 19.5, 26.4, 31.7, 19.0,
19.9, 18.2, 21.2, 13.6, 27.1, 21.7, 28.6, 24.0, 44.8, 31.1, 18.9, 20.4, 29.1, 5.6, 17.5, 39.8, 12.0, 33.2, 50.0, 20.1, 21.5, 28.5, 50.0, 22.0, 31.5, 22.
6, 50.0, 7.0, 15.3, 21.9, 20.7, 23.1, 22.2, 11.9, 13.4, 21.6, 19.7, 23.0, 8.4, 21.7, 20.6, 23.3, 19.6, 28.0, 8.5, 25.2, 14.2, 29.0, 20.1, 23.6, 19.0, 2
5.0, 20.3, 23.0]
```

[56]:
```
final_rmse
```

[56]:
```
2.6821354201018073
```

Making tableau dashboard



Obtaining the comparison in dashboard