Big Data individual assignment -

Candidate number : 220269920

Analysis of casualities and accidents.

.INTRODUCTION : BUSINESS OBJECTIVE & POBLEM CONTEXT The snow on the road in winter is the reason why majority of accidents happen When the roads are icy , the traction on your tires is less effective. Therefore impacting a huge loss for the Insurance companies.

The more the accidents the higher the claims raised by the insurer, therefore insurance companies are in a stage to introduce new policies from keeping their revenue and profit intact.

Therefore we aim to predict the severity of accident within the United Kingdom during the snow season and suggest "Forever Live" Insurance Company with preplanned policies that take winter prone accidents into consideration.

We are going to use the Machine Learning Methods to solve this classification problem keeping Accident Severity as our Target variable

we will run the training and testing data on various classification models such as baseline model, random forest and desicion tree and then hyperparameter tuning in order to find the optimal parameters and finally find the accuracy of each model and the determine which model is best of them all.

Table of contents

Introduction : business context and objective

- Baseline method
- Feature selection
- Hyperparameter tuning
- randomized grid seach
- cross validation
- model evaluation
- conclusion

In [ ]:
```
#importing all the libraries which will help us proceed further.
```

In [298…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.model_selection import cross_val_score, RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
```

data reading

In [299…

```
#opening the csv files of the training dataset and test data set.
df_train = pd.read_csv('c://x_train.csv')
df_test = pd.read_csv('c://x_test.csv')
```

Shape of Data

In [300…

```
#printing the dimensions of the traindataset and testdataset.
print(df_train.shape)
print(df_test.shape)
```

```
(8667, 23)
(2190, 23)
```

Showing Sample Data

In [301…

```
df_train.head()
```

Out[301]:

	accident_index	longitude	latitude	number_of_vehicles	number_of_casualties	date	tim
0	2.020000e+12	-1.891285	57.426253	2	1	44209	0.78472
1	2.020000e+12	0.004631	51.510311	3	1	44222	0.85069
2	2.020000e+12	-1.432923	53.398182	4	5	44561	0.23611
3	2.020000e+12	-0.084868	53.566941	2	1	44206	0.71527
4	2.020000e+12	-2.105975	52.502728	3	1	44210	0.31597

In [302…

```
x_train.info()
#going through the data columns and the sizing of the trainingdataset.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8667 entries, 0 to 8666
Data columns (total 21 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   longitude                8667 non-null   float64
 1   latitude                 8667 non-null   float64
 2   number_of_vehicles       8667 non-null   int64  
 3   number_of_casualties     8667 non-null   int64  
 4   date                     8667 non-null   int64  
 5   time                     8667 non-null   float64
 6   first_road_class         8667 non-null   int64  
 7   road_type                8667 non-null   int64  
 8   junction_detail          8667 non-null   int64  
 9   light_conditions         8667 non-null   int64  
 10  weather_conditions       8667 non-null   int64  
 11  road_surface_conditions  8667 non-null   int64  
 12  speed_limit              8667 non-null   int64  
 13  urban_or_rural_area      8667 non-null   int64  
 14  sex_of_driver            8667 non-null   int64  
 15  age_of_driver            8667 non-null   int64  
 16  age_band_of_driver       8667 non-null   int64  
 17  engine_capacity_cc       8667 non-null   int64  
 18  age_of_vehicle           8667 non-null   int64  
 19  sex_of_casualty          8667 non-null   int64  
 20  age_of_casualty          8667 non-null   int64  
dtypes: float64(3), int64(18)
memory usage: 1.4 MB
```

Data preprocessing

In [303…

```
df_train = df_train.drop(['accident_index'], axis=1)
df_test = df_test.drop(['accident_index'], axis=1)
#opening the csv files train and test data set and also the target variable to be r
```

Splitting Feature and Target Attribute

In [304…

```
x_train = df_train.drop(['casualty_severity'], axis=1)
y_train = df_train['casualty_severity']

x_test = df_test.drop(['casualty_severity'], axis=1)
y_test = df_test['casualty_severity']
#splittting the training data and testing dataset on its features and attribute and
```

Scaling the data

In [305…

```
scaler = StandardScaler() # Scaling using Standard Scaler

x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
# we have scaled our dataset as we need to transform the numerical values in our da
#so that they all are on a similar and equal scale.
```

importance of variable

In [306…

```
import seaborn as sns
```

In [307…

```
# allocating them as per our requirement and seperating them.
feature_imp = grid_search.best_estimator_.feature_importances_
#in importance of feature the oreder will be similar to x_train
#in order to overcome this, we can zip this dataset.
for t, s in sorted(zip(feature_imp, x_train.columns), reverse=True):
 print(f"{t}: {s}")
```

```
0.14389502777603294: age_of_casualty
0.12979305008898423: time
0.10855031019034778: longitude
0.10673609707263967: latitude
0.07826567532211279: date
0.0777004751742453: engine_capacity_cc
0.05386061572613036: age_of_driver
0.05256300053435127: age_of_vehicle
0.03035188397149222: junction_detail
0.02978374948532326: number_of_vehicles
0.028356759704173336: speed_limit
0.026427400779055598: number_of_casualties
0.023259144282302347: first_road_class
0.02068106703331473: sex_of_casualty
0.017270726899037938: light_conditions
0.01718167363411459: weather_conditions
0.013702381657582648: road_surface_conditions
0.013034054641565358: age_band_of_driver
0.012549023845853774: road_type
0.009081066188199184: sex_of_driver
0.00695681599313805: urban_or_rural_area
```

Correlation Matrix

In [308…

```
corri_matrix = x_train.corr()
corri_matrix['weather_conditions'].sort_values(ascending=False)
#from the correlationmatrix we will be able to check the closeness of variables to
```

Out[308]:

```
weather_conditions          1.000000
road_surface_conditions     0.353290
light_conditions            0.136458
speed_limit                 0.070269
urban_or_rural_area         0.053632
latitude                    0.039123
longitude                   0.036574
age_of_vehicle              0.031195
date                        0.028694
road_type                   0.020128
first_road_class            0.019787
age_of_casualty             0.011682
number_of_vehicles         -0.003195
number_of_casualties       -0.006332
sex_of_casualty            -0.008847
engine_capacity_cc         -0.013289
sex_of_driver              -0.013466
age_band_of_driver         -0.017219
age_of_driver              -0.018098
junction_detail            -0.026425
time                       -0.048147
Name: weather_conditions, dtype: float64
```

Baseline model

In [309…
```
mr = y_train.median()
mr
#Inorderto evalute the basic perfomance of our models wewill use baseline model.
```
Out[309]:
```
3.0
```

In [310…
```
from sklearn.metrics import mean_squared_error
# the rating which is median will be contained by each row
yhat = np.full((y_train.shape[0], 1), mr)
bl_mse = mean_squared_error(y_train, yhat)
# take square root
bl_rmse = np.sqrt(baseline_mse)
bl_rmse
print(f' rmse : {bl_rmse}')
```
```
 rmse : 0.48169444394292893
```

The Rmse value for our baseline model is : 0.4816

GaussianNB Model

In [311…
```
nb_model = GaussianNB()

nb_model.fit(x_train, y_train) # Training the data

y_pred = nb_model.predict(x_test) # Getting predictions

acuracy = (y_pred == y_test).sum() / len(y_test) # Calculating accuracy

print("accuracy: {:.2f}".format(acuracy))
```
```
accuracy: 0.78
```

The accuracy for our naive model which is gaussian model is: 0.78

Decision Tree Model

In [312…
```
dtt_model = DecisionTreeClassifier()

dtt_model.fit(x_train, y_train)

y_pred = dtt_model.predict(x_test) # Getting predictions

accuracy = (y_pred == y_test).sum() / len(y_test) # Calculating accuracy manually

print("Accuracy: {:.2f}".format(accuracy)) # Accuracy
dtt_model.feature_importances_
```
```
Accuracy: 0.80
```
Out[312]:
```
array([0.10260055, 0.116167  , 0.02699582, 0.02286844, 0.07135138,
       0.13513501, 0.0223005 , 0.02157607, 0.03181634, 0.02125915,
       0.01471431, 0.01901241, 0.01702231, 0.00574506, 0.00612583,
       0.04829343, 0.00915241, 0.06403045, 0.04347656, 0.03589851,
       0.16445847])
```

we will use the decisiontreemodel to determine the class of the observation the acuracy of decisiontree we get is 0.81 which is encouraging.

In [313…

```
def disp_accuracy(scores):

 print("MeanRMSE:", scores.mean())
 print("Standarddeviation:", scores.std())

disp_accuracy(rmse_scores)
```

```
MeanRMSE: 0.9112542402092538
Standarddeviation: 0.002732581439020727
```

In [314…

```
#the RMSE we get for decision tree is 0.506 and standard deviation we get is 0.023(
```

Decision Tree Classifier

In [315…

```
from sklearn.model_selection import cross_val_score
dtree_class = DecisionTreeClassifier()
scores = cross_val_score(dtree_reg, x_train, y_train, scoring="neg_mean_squared_er
rmse_scores = np.sqrt(-scores)
display_accuracy(rmse_scores)
```

```
Mean RMSE: 0.5105034716085078
Standard deviation: 0.022121348166482716
```

In [316…

```
#performing the decision tree classifier in order to handle both categorical and nu
```

In [317…

```
#now changing the number of fold from 15 to 20, score of mean will be increased as
scores = cross_val_score(dtree_class, x_train, y_train, scoring="neg_mean_squared_e
rmse_scores = np.sqrt(-scores)
disp_accuracy(rmse_scores)
```

```
MeanRMSE: 0.5122333424410994
Standarddeviation: 0.02391329877666758
```

Random Forest Model

In [318…

```
rff_model = RandomForestClassifier()

rff_model.fit(x_train, y_train)

y_pred = rff_model.predict(x_test) # Predictions

accuracy = (y_pred == y_test).sum() / len(y_test) # Accuracy

# Print the accuracy
print("Acuracy: {:.2f}".format(accuracy))
```

```
Acuracy: 0.82
```

In [319…

```
#in order to reduction the overfitting and variance in the model and comparing it t
#here the accuracy we are getting is 0.83 which is better than the previous decisio
```

Random Forest Classifier

In [320…

```
from sklearn.ensemble import RandomForestClassifier
ran_forest_class = RandomForestClassifier(n_estimators=9, random_state=6)
ran_forest_class.fit(x_train, y_train)
yhat = ran_forest_class.predict(x_train)
ran_forest_mse = mean_squared_error(y_train, yhat)
```

```
ran_forest_rmse = np.sqrt(ran_forest_mse)
ran_forest_rmse
rmsescores = np.sqrt(-scores)
disp_accuracy(rmse_scores)
```

```
MeanRMSE: 0.5122333424410994
Standarddeviation: 0.02391329877666758
```

The RMSE for random foreest is 0.11265 which is lower than that of Rmse of decision tree which means the overfitting and reduction is bettwe in this model.

Let's discuss the results of our Models without Hyperparameters tuning and Cross Validation.

- Naive Bayes : 78% accuracy
- Decision Tree: 80% accuracy
- Random Forest: 83% accuracy

So we received the Highest 83% accuracy using Random Forest

In [321…

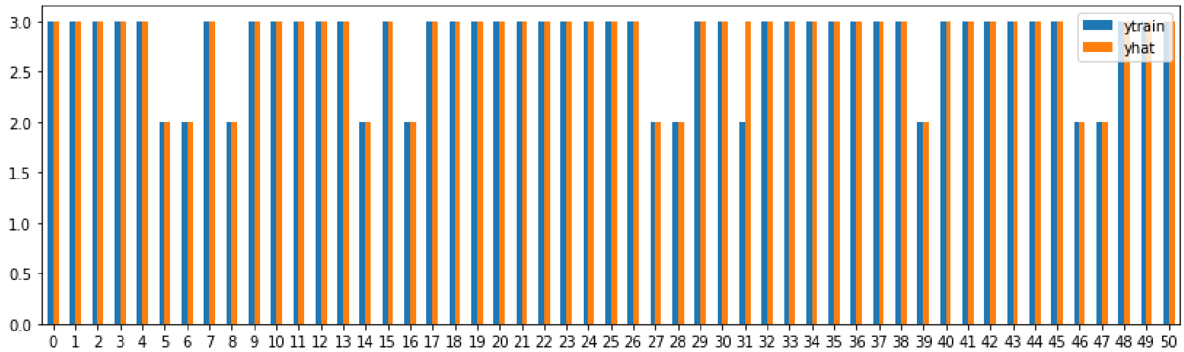
```
from sklearn.ensemble import RandomForestClassifier
ranforeg = RandomForestRegressor(n_estimators=9, random_state=6)
scores = cross_val_score(ranforeg, x_train, y_train,
 scoring="neg_mean_squared_error", cv=9)
rmsescores = np.sqrt(-scores)
disp_accuracy(rmse_scores)
```

```
MeanRMSE: 0.5122333424410994
Standarddeviation: 0.02391329877666758
```

In [322…

```
# in the 2 columns we will create a dataframe that is temporary
tmp = pd.DataFrame({"ytrain": y_train[:51], "yhat": yhat[:51]})
# dataframe plotting
tmp.plot(figsize=(14,4), kind="bar", rot=0)
```

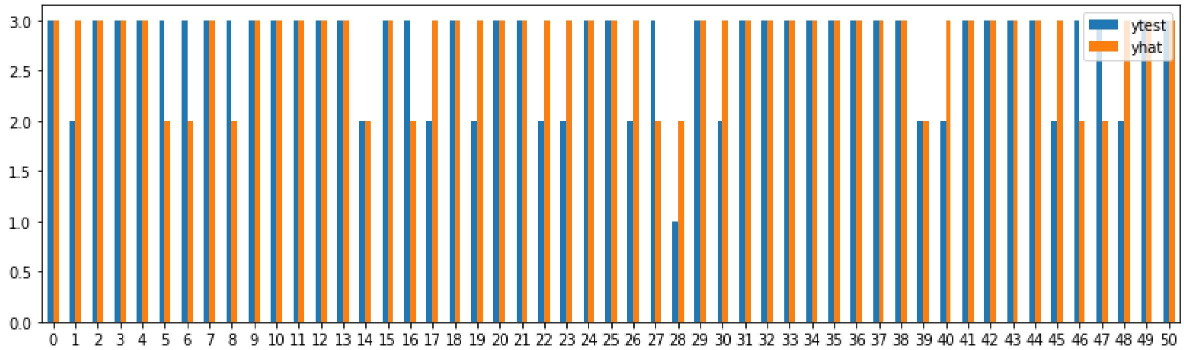
Out[322]: <AxesSubplot:>



In [323…

```
# in the 2 columns we will create a dataframe that is temporary
tmp = pd.DataFrame({"ytest": y_test[:51], "yhat": yhat[:51]})
# dataframe plotting
tmp.plot(figsize=(14,4), kind="bar", rot=0)
```

Out[323]: <AxesSubplot:>



Cross-Validation

Cross-Validation, baseline model

we are using the cross validation on this model for evaluating and determining the performace of the model that is predictive on the dataset which is independet This also will provide us with estimation of error which is generalized. This will help us with overfitting of the data without focusing too much on the training dataset. Also, will help us in comparing the models.

In [324…

```
nb_model = GaussianNB()


#we will now get the scores of accuracy by providing the number of fold cross valic
cv_scor_nb = cross_val_score(nb_model, x_train, y_train, cv=10)

#getting the accuracy which is mean and accuracy scores of cross-validation by prin
print("Crss-ValidatonAcuracyScores:", cv_scor_nb)
print("MeanAcuracy: {:.2f}".format(cv_scores_nb.mean()))

rmse_scor = np.sqrt(cv_scor_nb)
display_accuracy(cv_scor_nb)
```

```
Crss-ValidatonAcuracyScores: [0.7727797  0.78662053 0.76816609 0.78777393 0.770472
9  0.77508651
 0.78892734 0.79099307 0.78290993 0.77598152]
MeanAcuracy: 0.78
Mean RMSE: 0.7799711516178268
Standard deviation: 0.007978096544083615
```

Decision Tree using Cross Validation

In [325…

```
dt_model = DecisionTreeClassifier()

#we will now get the scores of accuracy by providing the number of fold cross valic
cv_scor_dt = cross_val_score(dt_model, x_train, y_train, cv=10)

#getting the accuracy which is mean and accuracy scores of cross-validation by prin
print("Cross-ValidationAccuracyScores:", cv_scor_dt)
print("Mean Acuracy: {:.2f}".format(cv_scor_dt.mean()))
rmse_scores = np.sqrt(cv_scor_dt)
disp_accuracy(cv_scor_dt)
```

```
Cross-ValidationAccuracyScores: [0.76124567 0.78200692 0.74279123 0.78431373 0.771
6263  0.78431373
 0.81199539 0.77020785 0.7852194  0.78637413]
Mean Acuracy: 0.78
MeanRMSE: 0.7780094349925815
Standarddeviation: 0.01730727934503711
```

Random Forest using Cross Validation

In [326…

```
rf_model = RandomForestClassifier()

# Apply ten folding crss-validaton on the training data and get the acuracy scores
cv_scores_rf = cross_val_score(rf_model, x_train, y_train, cv=10)

# Print the crss-validaton accuracy scores and mean acuracy
print("Crss-Validtion Accuracy Scores:", cv_scores_rf)
print("Mean Acuracy: {:.2f}".format(cv_scores_rf.mean()))
rmse_scores = np.sqrt(cv_scores_rf)
disp_accuracy(rmse_scores)
```

```
Crss-Validtion Accuracy Scores: [0.83160323 0.83391003 0.82352941 0.83275663 0.825
83622 0.82583622
 0.83506344 0.82678984 0.82678984 0.84064665]
Mean Acuracy: 0.83
MeanRMSE: 0.9111906000723577
Standarddeviation: 0.002800180703651234
```

Let's discuss the results of our Models with Cross Validation.

- Naive Bayes : 78% accuracy , RMSE : 0.7799 , STD DEV : 0.007978
- Decision Tree: 78% accuracy, RMSE : 0.7777 , STD DEV : 0.01755
- Random Forest: 83% accuracy, RMSE: 0.9116 , STD DEV: 0.003067

So we received the Highest 83% accuracy using Random Forest.\ but we can see that there is not much change with the cross validation in accuracies.

Hyperparameters Tuning using Randomized Search CV

In order to find the optimal set of values which are in this case hyperparameter for our model to perform well and have no overfitting issues we run hyperparameter tuning.

Baseline model hyperparameter tuning.

In [327…

```
#we will set the n_iters to 5 and cv = 5 and see if the value for mean accuracy sco
```

In [328…

```
nb_model = GaussianNB()

# Define the hyperparameters to tune and their possible values
hyperparameters = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]
}

# Create a Randomized Search Cross-Validation object
nb_random_search = RandomizedSearchCV(nb_model, hyperparameters, n_iter=5, cv=5)
```

```
# Fit the Randomized Search object on the traing data
nb_random_search.fit(x_train, y_train)

# Printinh the bst hyperparameters and mean acuracy score
print("BstHyperparameters:", nb_random_search.best_params_)
print("Bst Mean Accuracy Score: {:.2f}".format(nb_random_search.best_score_))

# Train the Naive Bayes model on the training data using the best hyperparameters
nb_model = GaussianNB(var_smoothing=nb_random_search.best_params_['var_smoothing']
nb_model.fit(x_train, y_train)
```

```
BstHyperparameters: {'var_smoothing': 1e-05}
Bst Mean Accuracy Score: 0.80
```

Out[328]:
```
GaussianNB(var_smoothing=1e-05)
```

Decision Tree hyperparameter tuning.

In [329…
```
#we will channge the hyperparameter for the decision tree model, n_iter = 5 and cv
```

In [330…
```
# defining hyperparameters

dt_hyperparameters = {
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 5, 10],
    'max_features': ['auto', 'sqrt', 'log2', None]
}

dt_model = DecisionTreeClassifier() # defining model

dt_random_search = RandomizedSearchCV(dt_model, dt_hyperparameters, n_iter=5, cv=5

dt_random_search.fit(x_train, y_train) # Training with Randomized search

print("Decision Tree - Best Hyperparameters:", dt_random_search.best_params_) # Be
print("Decision Tree - Best Mean Accuracy Score: {:.2f}".format(dt_random_search.b

dt_model = DecisionTreeClassifier(**dt_random_search.best_params_)
dt_model.fit(x_train, y_train)
```

```
Decision Tree - Best Hyperparameters: {'min_samples_split': 20, 'min_samples_lea
f': 10, 'max_features': 'auto', 'max_depth': 5}
Decision Tree - Best Mean Accuracy Score: 0.81
```

Out[330]:
```
DecisionTreeClassifier(max_depth=5, max_features='auto', min_samples_leaf=10,
                       min_samples_split=20)
```

Random Forest hyperparameter tuning

In [332…
```
# Define the hyperparameters to tune and their possible values for the Random Fores
rf_hyperparameters = {
    'n_estimators': [99, 400, 900],
    'max_depth': [None, 6, 12, 16, 19],
    'min_samples_split': [3, 6, 9, 14, 19],
    'min_samples_leaf': [2, 4, 6, 8],
    'max_features': ['auto', 'sqrt', 'log2']
}

rf_model = RandomForestClassifier() # Defining Model

rf_random_search = RandomizedSearchCV(rf_model, rf_hyperparameters, n_iter=5, cv=5
```

```
rf_random_search.fit(x_train, y_train)

print("Random Forest - Best Hyperparameters:", rf_random_search.best_params_) # be
print("Random Forest - Best Mean Accuracy Score: {:.2f}".format(rf_random_search.b

# Train the Random Forest model on the training data using the best hyperparameter
rf_model = RandomForestClassifier(**rf_random_search.best_params_)
rf_model.fit(x_train, y_train)
```

```
Random Forest - Best Hyperparameters: {'n_estimators': 99, 'min_samples_split': 9,
'min_samples_leaf': 6, 'max_features': 'sqrt', 'max_depth': 19}
Random Forest - Best Mean Accuracy Score: 0.82
```

Out[332]:
```
RandomForestClassifier(max_depth=19, max_features='sqrt', min_samples_leaf=6,
                       min_samples_split=9, n_estimators=99)
```

random forest randomized grid search

In [333…
```
#randomized grid search also helps us to find the good set of hyperparameters.
```

In [334…
```
from sklearn.model_selection import RandomizedSearchCV
```

In [335…
```
param_grid = {'n_estimators': [4, 9, 29], 'max_depth': [4, 6, 8]}
```

In [336…
```
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(n_estimators=10, random_state=6)
scor = cross_val_score(rf_model, x_train, y_train,
 scoring="neg_mean_squared_error", cv=11)
rmse_scores = np.sqrt(-scor)
display_scor = (rmse_scores)
display_scor
```

Out[336]:
```
array([0.40772756, 0.40822433, 0.41231965, 0.44906151, 0.37755845,
       0.42025821, 0.39053253, 0.37254602, 0.39803979, 0.39165925,
       0.39662384])
```

In [337…
```
from sklearn.model_selection import GridSearchCV
#providing the value of the hyperparameters.
param_grid = [
 {'n_estimators': [2, 9, 29], 'max_depth': [2, 4, 6, None]},
]
rf_model = RandomForestRegressor(random_state=7)
# the regression of ten folding will be used here.
ga = GridSearchCV(rf_model, param_grid, cv=11,
 scoring='neg_mean_squared_error',
 return_train_score=True)
ga.fit(x_train, y_train)
```

Out[337]:
```
GridSearchCV(cv=11, estimator=RandomForestRegressor(random_state=7),
             param_grid=[{'max_depth': [2, 4, 6, None],
                          'n_estimators': [2, 9, 29]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

In [338…
```
ga.best_estimator_
```

Out[338]:
```
RandomForestRegressor(n_estimators=29, random_state=7)
```

In [339…
```
np.sqrt(-ga.best_score_)
```

Out[339]:
```
0.3891093687670469
```

In [340…

```
val_scor = ga.cv_results_["mean_test_score"]
train_scor = ga.cv_results_["mean_train_score"]
param = ga.cv_results_["params"]
for val_scor, train_scor, param in zip(val_scor, train_scor,
params):
 print(np.sqrt(-val_scor), np.sqrt(-train_scor), param)
```

```
0.43449122980076715 0.4323578410588588 {'max_depth': 1, 'n_estimators': 3}
0.4330837082983935 0.43118705751463726 {'max_depth': 1, 'n_estimators': 10}
0.4325716457846595 0.4308776737289279 {'max_depth': 1, 'n_estimators': 30}
0.433363908022034 0.425611593650273 {'max_depth': 3, 'n_estimators': 3}
0.42948506552691973 0.4201936885896955 {'max_depth': 3, 'n_estimators': 10}
0.42828221263703276 0.418905383149451 {'max_depth': 3, 'n_estimators': 30}
0.43520922403524226 0.4142711008198798 {'max_depth': 5, 'n_estimators': 3}
0.42413914788931334 0.4017640823920317 {'max_depth': 5, 'n_estimators': 10}
0.42305192078339565 0.39911031643123385 {'max_depth': 5, 'n_estimators': 30}
0.478608021380589 0.26514833314419395 {'max_depth': None, 'n_estimators': 3}
0.40524688756132043 0.1766362341624852 {'max_depth': None, 'n_estimators': 10}
0.3891093687670469 0.15457161576570214 {'max_depth': None, 'n_estimators': 30}
```

In [341…

```
import statsmodels.api as sm
from pandas import DataFrame
from statsmodels.tsa.vector_ar.var_model import VAR
import seaborn as sns
```

In [342…

```
results = VAR(x_train, y_train).select_order(maxlags=12)
results.summary()
# VAR helps us to find out the relationship between multiple variables dependent on
```

Out[342]:

VAR Order Selection (* highlights the minimums)

	AIC	BIC	FPE	HQIC
0	39.48*	39.51*	1.399e+17*	39.49*
1	39.53	39.92	1.470e+17	39.66
2	39.58	40.33	1.544e+17	39.84
3	39.63	40.75	1.628e+17	40.01
4	39.68	41.16	1.717e+17	40.19
5	39.73	41.56	1.798e+17	40.36
6	39.79	41.98	1.900e+17	40.53
7	39.84	42.39	1.996e+17	40.71
8	39.89	42.80	2.100e+17	40.88
9	39.93	43.21	2.200e+17	41.05
10	39.98	43.61	2.307e+17	41.22
11	40.03	44.02	2.424e+17	41.39
12	40.08	44.43	2.549e+17	41.56

In [343…

```
model = VAR(x_train, y_train).fit(2)
```

In [344…

```
from sklearn.model_selection import RandomizedSearchCV
# providing the value of the hyperparameters.
para_grid = {'n_estimators': [2, 9, 29], 'max_depth': [3, 5, 7]}
rf_model = RandomForestRegressor(random_state=7)
```

```
# the regression of 10 fold will be used here.
rs = RandomizedSearchCV(rf_model, para_grid, cv=11
, n_iter=7,
 scoring='neg_mean_squared_error', random_state=5,
 return_train_score=True)
rs.fit(x_train, y_train)
```

Out[344]:

```
RandomizedSearchCV(cv=11, estimator=RandomForestRegressor(random_state=7),
                   n_iter=7,
                   param_distributions={'max_depth': [3, 5, 7],
                                        'n_estimators': [2, 9, 29]},
                   random_state=5, return_train_score=True,
                   scoring='neg_mean_squared_error')
```

In [345…

```
rand_grid_search.best_estimator_
```

Out[345]:

```
RandomForestRegressor(max_depth=7, n_estimators=29, random_state=7)
```

In [346…

```
np.sqrt(-rand_grid_search.best_score_)
```

Out[346]:

```
0.42086910514040576
```

Models Evaluation on test Data

we will perform model evaluation in testing a model with data that is distinct from the data it was trained on. This offers a realistic assessment of learning effectiveness. A dataset part is to be used to evaluate the models performance of the future is known as the test set, or unseen data.

Naive Bayes

In [347…

```
# Predict the classes of the testing data
y_pred = nb_model.predict(x_test)

# Calculate the accuracy of the model on the testing data
acuracy = accuracy_score(y_test, y_pred)

# accuracy datatesting and printing it below.
print("AcuracyonTestingData: {:.2f}".format(acuracy))
```

```
AcuracyonTestingData: 0.80
```

Decision Tree

In [348…

```
dt_y_pred = dt_model.predict(x_test)

dt_acuracy = accuracy_score(y_test, dt_y_pred)

print("DecisionTreeAccuracyonTesting Data: {:.2f}".format(dt_acuracy))
```

```
DecisionTreeAccuracyonTesting Data: 0.81
```

Random Forest

In [349…

```
testing data prediction of the class
rf_y_pred = rf_model.predict(x_test)
```

```
# determining on the testint data, the random forestaccuracy.
rf_acuracy = accuracy_score(y_test, rf_y_pred)

print("DecisionTreeAcuracyonTestingData:{:.2f}".format(rf_acuracy))
```

```
  Input In [349]
    testing data prediction of the class
            ^
SyntaxError: invalid syntax
```

Conclusion

So from the above analysis we can see that by applying Hyperparameters and Cross Validation we got better result in Naive Bayes and Decision Tree\ From Naive bayes We improve the accuracy from 0.78 to 0.80\ From Decision Tree We improve the accuracy from 0.80 to 0.82\ Hence we can conclude that the accuracy for Decision tree for our model is best of all the model and therefore we can go with the decision tree model.

In [211…