A PROJECT REPORT

on

"Comparative Analysis of Machine Learning Algorithms"

Submitted to

KIIT Deemed to be University

In Partial Fulfillment of the Requirement for the Award of

BACHELOR'S DEGREE IN

COMPUTER SCIENCE & ENGINEERING

BY

AARYASH KUMAR	**21051871**
DHRUV KUMAR	**21052495**
A.SUBHAM PATRO	**21051021**
APURVA JHA	**21052483**
RISHAV RAJ	**21051419**

UNDER THE GUIDANCE OF

Dr. Jayanta Mondal

Project Configuration (HW/SW Specification and CODE)

Loan Prediction Model:

Hardware Requirement:

Hardware Requirement
RAM: 16 GB
SYSTEM TYPE: 64-bit Operating System

Software Requirement:

Operating System: Windows 11

Technical Specification

The technical tools used in making this project include the following:

Python3: Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Jupyter Notebook: The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

CODE:

Import libraries

```
import numpy as np
import pandas as pd
```

```
from sklearn.model_selection import train_test_split,RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.metrics import accuracy_score,roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
import matplotlib.pyplot as plt
!pip install xgboost
import xgboost as xgb
from xgboost import XGBClassifier
```

Load Dataset

```
loan_train =pd.read_csv('train_csv.csv' )
print(loan_train.shape) # (614, 13)
loan_train.head()
```

EDA

```
total_null=loan_train.isnull().sum().sort_values(ascending=False)
total_null.head(10)
loan_train['Gender'] = loan_train ['Gender'].fillna(
loan_train['Gender'].dropna ().mode().values[0])
loan_train['Married'] = loan_train['Married'].fillna(
loan_train['Married'].dropna ().mode().values[0])
loan_train['Dependents'] =
loan_train['Dependents'].fillna(loan_train['Dependents'].dropna
().mode().values[0])
loan_train['Self Employed'] = loan_train[
'Self_Employed'].fillna(loan_train['Self_Employed'].dropna().mode().values
[0] )
loan_train['LoanAmount'] = loan_train['LoanAmount'].fillna(loan_train
['LoanAmount'].dropna().mean())
loan_train['Loan_Amount_Term'] = loan_train[
'Loan_Amount_Term'].fillna(loan_train['Loan_Amount_Term'].dropna
().mode().values [0])
loan_train['Credit_History'] = loan_train [
'Credit_History'].fillna(loan_train['Credit_History'].dropna().mode().valu
es [0])
loan_train.info()
print(set(loan_train['Gender'].values.tolist()))
print(set(loan_train['Dependents'].values.tolist()))
print(set(loan_train['Married'].values.tolist()))
print(set(loan_train['Education'].values.tolist()))
print(set(loan_train['Self_Employed'].values.tolist()))
print(set(loan_train['Loan_Status'].values.tolist()))
print(set(loan_train['Property_Area'].values.tolist()))
```

Train Dataset

```
loan_train['Loan_Status'] = loan_train['Loan_Status'].map({'N': 0, 'Y':
1}).fillna(0).astype(int)
loan_train = pd.get_dummies(loan_train, columns=['Gender', 'Dependents',
'Married', 'Education', 'Self Employed', 'Property_Area'])
standardScaler = StandardScaler()
loan_train['Loan_Term'] = loan_train['Loan_Amount_Term']
del loan_train['Loan_Amount_Term']
loan_train[['CoapplicantIncome', 'LoanAmount', 'Loan_Term']] =
standardScaler.fit_transform(loan_train[['CoapplicantIncome',
'LoanAmount', 'Loan_Term']])
y = loan_train['Loan_Status']  # Select target variable (Loan_Status)
X = loan_train.drop(['Loan_Status', 'Loan_ID'], axis=1)  # Drop target and
ID columns
# Split data using the defined variables
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Tuning models and test for all features

XGBoost

```
gbm_param_grid = {
    'n_estimators': range(1, 1000, 10),
    'max_depth': range(1, 20),
    'learning_rate': [.1, .4, .45, .5, .55, .6],
    'colsample_bytree': [.6, .7, .8, .9, 1.0, 1.1]
}
X_train = pd.get_dummies(X_train, columns=['Self_Employed'])  # One-hot
encode Self_Employed
X_test = pd.get_dummies(X_test, columns=['Self_Employed'])
xgb_classifier = XGBClassifier(enable_categorical=True)
xgb_random = RandomizedSearchCV(param_distributions=gbm_param_grid,
                                estimator=xgb_classifier,
scoring="accuracy",
                                verbose=0, n_iter=100, cv=4)
error_score='raise'
xgb_random.fit(X_train,y_train)
print(f'Best parameters: {xgb_random.best_params_}')
y_pred = xgb_random.predict(X_test)
print(f'Accuracy: {np.sum(y_pred == y_test) / len(y_test)}')
```

Decision Tree

```
param_grid = {
'max_depth': range(4,25),
'min_samples_leaf': range(10, 100, 10),
```

```
'min_samples_split' : range(10, 100, 10),
'criterion': ['gini', 'entropy']
}
n_folds = 5
dt = DecisionTreeClassifier (random_state=np.random.randint(0, 100))
dt_grid = GridSearchCV(dt, param_grid, cv = n_folds,
return_train_score=True, verbose=0)
dt_grid.fit(X_train,y_train)
print(dt_grid.best_params_)
# {'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 20,
'min_samples_split': 10}
y_pred_best=dt_grid.predict(X_test)
acc = metrics.accuracy_score (y_test, y_pred_best)
print(acc)
```

Random Forest

```
rf_param_grid = {
'max_depth': range(4,25),
'min_samples_leaf': range(10, 100, 10),
'min_samples_split' : range(10, 100, 10),
'criterion': ['gini', 'entropy']
}
rf RandomForestClassifier()
rf_random = Randomized SearchCV(param_distributions=rf_param_grid,
estimator = rf, scoring = "accuracy", verbose = 0, n_iter = 100, cv = 4)
rf_random.fit(X_train,y_train)
best_params = rf_random.best_params_
print(f'Best parameters: {best_params}')
#Best parameters: {'n_estimators': 101}
y_pred1 = rf_random.predict(X_test)
print(f'Accuracy: {np.sum(y_pred1==y_test)/len(y_test)}')
```

Support Vector

```
svm_param_grid = {
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'C': range(1, 11)
}
# Create an SVC model
svm = SVC()
# Perform randomized search with cross-validation
svm_random = RandomizedSearchCV(param_distributions=svm_param_grid,
                                estimator=svm, scoring="accuracy",
                                verbose=0, n_iter=40, cv=4)
# Train the model on your training data
svm_random.fit(X_train, y_train)
# Get the best parameters found during tuning
best_params = svm_random.best_params_
```

```
print(f'Best parameters: {best_params}')
# Predict on the test data using the best model
y_pred_best = svm_random.predict(X_test)
# Calculate accuracy on the test data
acc = metrics.accuracy_score(y_test, y_pred_best)
print(acc)
```

CAR PRICE PREDICTION:

Hardware Requirement:

Hardware Requirement
RAM: 16 GB
SYSTEM TYPE: 64-bit Operating System

Software Requirement:

Operating System: Windows 10

Technical Specification

The technical tools used in making this project include the following:

Python3: Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Google Colab: The Google Colab is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

CODE:

1. Import libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# preprocessing
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score,
GridSearchCV, StratifiedKFold
import pandas_profiling as pp

# models
from sklearn.linear_model import LinearRegression, SGDRegressor, RidgeCV
from sklearn.svm import SVR, LinearSVR
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor, ExtraTreesRegressor
from sklearn.ensemble import BaggingRegressor, AdaBoostRegressor,
VotingRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.tree import DecisionTreeRegressor
import sklearn.model_selection
from sklearn.model_selection import cross_val_predict as cvp
from sklearn import metrics
from sklearn.metrics import r2_score, mean_absolute_error,
mean_squared_error
import xgboost as xgb
import lightgbm as lgb

# model tuning
from hyperopt import STATUS_OK, Trials, fmin, hp, tpe, space_eval

import warnings
warnings.filterwarnings("ignore")
```

2. Download datasets

```
train0 =
pd.read_csv('/kaggle/input/craigslist-carstrucks-data/craigslistVehicles.c
sv')
train0.head(5)
drop_columns = ['url', 'city', 'city_url', 'make', 'title_status', 'VIN',
'size', 'image_url', 'desc', 'lat','long']
train0 = train0.drop(columns = drop_columns)
```

```
train0.info()
train0 = train0.dropna()
train0.head(5)
# from the my kernel:
https://www.kaggle.com/vbmokin/automatic-selection-from-20-classifier-mode
ls
# Determination categorical features
numerics = ['int8', 'int16', 'int32', 'int64', 'float16', 'float32',
'float64']
categorical_columns = []
features = train0.columns.values.tolist()
for col in features:
    if train0[col].dtype in numerics: continue
    categorical_columns.append(col)
# Encoding categorical features
for col in categorical_columns:
    if col in train0.columns:
        le = LabelEncoder()
        le.fit(list(train0[col].astype(str).values))
        train0[col] = le.transform(list(train0[col].astype(str).values))
train0['year'] = (train0['year']-1900).astype(int)
train0['odometer'] = train0['odometer'].astype(int)
train0.head(10)
train0.info()
```

3. EDA

```
train0['price'].value_counts()
train0 = train0[train0['price'] > 1000]
train0 = train0[train0['price'] < 40000]
# Rounded ['odometer'] to 5000
train0['odometer'] = train0['odometer'] // 5000
train0 = train0[train0['year'] > 110]
train0.info()
train0.corr()
train0.describe()
pp.ProfileReport(train0)
```

4. Preparing to modeling

```
target_name = 'price'
```

```
train_target0 = train0[target_name]
train0 = train0.drop([target_name], axis=1)
# Synthesis test0 from train0
train0, test0, train_target0, test_target0 = train_test_split(train0,
train_target0, test_size=0.2, random_state=0)
# For boosting model
train0b = train0
train_target0b = train_target0
# Synthesis valid as test for selection models
trainb, testb, targetb, target_testb = train_test_split(train0b,
train_target0b, test_size=valid_part, random_state=0)
#For models from Sklearn
scaler = StandardScaler()
train0 = pd.DataFrame(scaler.fit_transform(train0), columns =
train0.columns)
train0.head(3)
len(train0)
# Synthesis valid as test for selection models
train, test, target, target_test = train_test_split(train0, train_target0,
test_size=valid_part, random_state=0)
train.head(3)
test.head(3)
train.info()
test.info()
acc_train_r2 = []
acc_test_r2 = []
acc_train_d = []
acc_test_d = []
acc_train_rmse = []
acc_test_rmse = []
def acc_d(y_meas, y_pred):
    # Relative error between predicted y_pred and measured y_meas values
    return mean_absolute_error(y_meas,
y_pred)*len(y_meas)/sum(abs(y_meas))

def acc_rmse(y_meas, y_pred):
    # RMSE between predicted y_pred and measured y_meas values
    return (mean_squared_error(y_meas, y_pred))**0.5
def acc_boosting_model(num,model,train,test,num_iteration=0):
    # Calculation of accuracy of boosting model by different metrics
```

```
    global acc_train_r2, acc_test_r2, acc_train_d, acc_test_d,
acc_train_rmse, acc_test_rmse

    if num_iteration > 0:
        ytrain = model.predict(train, num_iteration = num_iteration)
        ytest = model.predict(test, num_iteration = num_iteration)
    else:
        ytrain = model.predict(train)
        ytest = model.predict(test)

    print('target = ', targetb[:5].values)
    print('ytrain = ', ytrain[:5])

    acc_train_r2_num = round(r2_score(targetb, ytrain) * 100, 2)
    print('acc(r2_score) for train =', acc_train_r2_num)
    acc_train_r2.insert(num, acc_train_r2_num)

    acc_train_d_num = round(acc_d(targetb, ytrain) * 100, 2)
    print('acc(relative error) for train =', acc_train_d_num)
    acc_train_d.insert(num, acc_train_d_num)

    acc_train_rmse_num = round(acc_rmse(targetb, ytrain) * 100, 2)
    print('acc(rmse) for train =', acc_train_rmse_num)
    acc_train_rmse.insert(num, acc_train_rmse_num)

    print('target_test =', target_testb[:5].values)
    print('ytest =', ytest[:5])

    acc_test_r2_num = round(r2_score(target_testb, ytest) * 100, 2)
    print('acc(r2_score) for test =', acc_test_r2_num)
    acc_test_r2.insert(num, acc_test_r2_num)

    acc_test_d_num = round(acc_d(target_testb, ytest) * 100, 2)
    print('acc(relative error) for test =', acc_test_d_num)
    acc_test_d.insert(num, acc_test_d_num)

    acc_test_rmse_num = round(acc_rmse(target_testb, ytest) * 100, 2)
    print('acc(rmse) for test =', acc_test_rmse_num)
    acc_test_rmse.insert(num, acc_test_rmse_num)
```

```
def acc_model(num,model,train,test):
    # Calculation of accuracy of model акщь Sklearn by different metrics

    global acc_train_r2, acc_test_r2, acc_train_d, acc_test_d,
acc_train_rmse, acc_test_rmse

    ytrain = model.predict(train)
    ytest = model.predict(test)

    print('target = ', target[:5].values)
    print('ytrain = ', ytrain[:5])

    acc_train_r2_num = round(r2_score(target, ytrain) * 100, 2)
    print('acc(r2_score) for train =', acc_train_r2_num)
    acc_train_r2.insert(num, acc_train_r2_num)

    acc_train_d_num = round(acc_d(target, ytrain) * 100, 2)
    print('acc(relative error) for train =', acc_train_d_num)
    acc_train_d.insert(num, acc_train_d_num)

    acc_train_rmse_num = round(acc_rmse(target, ytrain) * 100, 2)
    print('acc(rmse) for train =', acc_train_rmse_num)
    acc_train_rmse.insert(num, acc_train_rmse_num)

    print('target_test =', target_test[:5].values)
    print('ytest =', ytest[:5])

    acc_test_r2_num = round(r2_score(target_test, ytest) * 100, 2)
    print('acc(r2_score) for test =', acc_test_r2_num)
    acc_test_r2.insert(num, acc_test_r2_num)

    acc_test_d_num = round(acc_d(target_test, ytest) * 100, 2)
    print('acc(relative error) for test =', acc_test_d_num)
    acc_test_d.insert(num, acc_test_d_num)

    acc_test_rmse_num = round(acc_rmse(target_test, ytest) * 100, 2)
    print('acc(rmse) for test =', acc_test_rmse_num)
    acc_test_rmse.insert(num, acc_test_rmse_num)
```

5. Tuning models and test for all features

```
# Support Vector Machines

svr = SVR()
svr.fit(train, target)
acc_model(1,svr,train,test)


# Decision Tree Regression


decision_tree = DecisionTreeRegressor()
decision_tree.fit(train, target)
acc_model(5,decision_tree,train,test)


# Random Forest


#random_forest = GridSearchCV(estimator=RandomForestRegressor(),
param_grid={'n_estimators': [100, 1000]}, cv=5)
random_forest = RandomForestRegressor()
random_forest.fit(train, target)
print(random_forest.best_params_)
acc_model(6,random_forest,train,test)


#XGBoost


xgb_clf = xgb.XGBRegressor({'objective': 'reg:squarederror'})
parameters = {'n_estimators': [60, 100, 120, 140],
'learning_rate': [0.01, 0.1],
'max_depth': [5, 7],
'reg_lambda': [0.5]}
xgb_reg = GridSearchCV(estimator=xgb_clf, param_grid=parameters, cv=5,
n_jobs=-1).fit(trainb, targetb)
print("Best score: %0.3f" % xgb_reg.best_score_)
```

```
print("Best parameters set:", xgb_reg.best_params_)
```

```
acc_boosting_model(7,xgb_reg,trainb,testb)
```

HOUSE PRICE PREDICTION:

Hardware Requirement:

Hardware Requirement
RAM: 16 GB
SYSTEM TYPE: 64-bit Operating System

Software Requirement:

Operating System: Windows 10

Technical Specification

The technical tools used in making this project include the following:

Python3: Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Google Colab: The Google Colab is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

CODE:

1. Import libraries

```
import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings("ignore")


from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import r2_score, mean_absolute_error,
mean_squared_error

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR

from xgboost import XGBRegressor

from sklearn.preprocessing import PolynomialFeatures
```

2. Download datasets

```
df=pd.read_csv("../input/house-prices-advanced-regression-techniques/train
.c v")
```

3. EDA

```
df.head()

df.shape
```

```
df.info()
df.describe().T
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), cmap="RdBu")
plt.title("Correlations Between Variables", size=15)
plt.show()
```

4. Preparing to modeling

```
X = df.drop("SalePrice", axis=1)
y = df["SalePrice"]
X = pd.get_dummies(X, columns=cat_cols)
important_num_cols.remove("SalePrice")
scaler = StandardScaler()
X[important_num_cols] = scaler.fit_transform(X[important_num_cols])
X.head()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

5. Tuning models and test for all features

```
#SVM

svr = SVR(C=100000)
svr.fit(X_train, y_train)
predictions = svr.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
```

```
print("-"*30)
rmse_cross_val = rmse_cv(svr)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "SVR","MAE": mae, "MSE": mse, "RMSE": rmse, "R2
Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)

# Random Forest

random_forest = RandomForestRegressor(n_estimators=100)
random_forest.fit(X_train, y_train)
predictions = random_forest.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(random_forest)
print("RMSE Cross-Validation:", rmse_cross_val
new_row = {"Model": "RandomForestRegressor","MAE": mae, "MSE": mse,
"RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)":
rmse_cross_val}
models = models.append(new_row, ignore_index=True)

#XGBoost

xgb = XGBRegressor(n_estimators=1000, learning_rate=0.01)
xgb.fit(X_train, y_train)
predictions = xgb.predict(X_test)
```

```
mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(xgb)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "XGBRegressor","MAE": mae, "MSE": mse, "RMSE": rmse,
"R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
```

Stock PRICE PREDICTION:

Hardware Requirement:

Hardware Requirement
RAM: 16 GB
SYSTEM TYPE: 64-bit Operating System

Software Requirement:

Operating System: Windows 10

Technical Specification

The technical tools used in making this project include the following:

Python3: Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Jupyter Notebook: The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

CODE:

Import libraries

```
import pandas as pd
import numpy as np
from sklearn import metrics
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
import xgboost as xgb
```

Load Dataset

```
df=pd.read_csv("GOOG.csv")
df.head()
```

EDA

```
df=df.drop(columns=[

'symbol','adjClose','adjHigh','adjLow','adjOpen','adjVolume','divCash
','splitFactor'
],axis=1)
df.head()
#Are there any Duplicate values
df.duplicated().sum().any()
# Cheaking & reviewing DataFrame information
df.isnull().values.any()
df.describe()
df['date'] = pd.to_datetime(df['date'])
df.head()
df['date'] = df['date'].dt.strftime('%Y-%m-%d')
df.head()
# Assuming df is your DataFrame and you want to drop non-numeric
columns before plotting
numeric_df = df.select_dtypes(include=['number'])
# Plot the correlation heatmap
plt.figure(figsize=(16, 8))
sns.heatmap(numeric_df.corr(), cmap="Blues", annot=True)
plt.show()
#showing visualization on all variables in data
sns.pairplot(df)
df['open'].hist()
df['high'].hist()
df['low'].hist()
df['close'].hist()
df['volume'].hist()
#Review box plots
f, axes = plt.subplots(1,4)
sns.boxplot(y='open', data=df, ax=axes[0])
```

```
sns.boxplot(y='high', data=df, ax=axes[1])
sns.boxplot(y='low', data=df, ax=axes[2])
sns.boxplot(y='close', data=df, ax=axes[3])
plt.tight_layout()
import plotly.graph_objects as go
figure =
go.Figure(data=[go.Candlestick(x=df["date"],open=df["open"],high=df["
high"],low=df["low"],close=df["close"])])
figure.update_layout(title= "Google Stock Price Analysis",
xaxis_rangeslider_visible=False)
figure.show()
```

Preparing to modeling

```
# Split the dataset
x=df[['open','high','low','volume']].values # independent variables
y=df['close'].values # dependent variable
from sklearn.model_selection import train_test_split
# Splitting the data 80% train and 20% testing
x_train,x_test,y_train,y_test=
train_test_split(x,y,test_size=0.2,random_state=0)
# Checking the shape for train data
print('Train:', x_train.shape)
print('test:', x_test.shape)
```

Tuning models and test for all features

Random Forest

```
# Assuming x_train and y_train are your training data and labels
rf = RandomForestRegressor()
# Fit the model to the training data
rf.fit(x_train,y_train)
yrf_pred=rf.predict(x_test)
print(yrf_pred)
x2=abs(yrf_pred-y_test)
y2=100*(x2/y_test)
accuracy=100-np.mean(y2)
print('Accuracy:',round(accuracy,2),'%.')
```

Decision Tree Regression

```
dtr=DecisionTreeRegressor()
dtr.fit(x_train,y_train)
ydtr_pred=dtr.predict(x_test)
print(ydtr_pred)
x3=abs(ydtr_pred-y_test)
```

```
y3=100*(x2/y_test)
accuracy=100-np.mean(y2)
print('Accuracy:',round(accuracy,2),'%.')
```

Support Vector Regression

```
Svr=SVR()
Svr.fit(x_train,y_train)
ysvr_pred=Svr.predict(x_test)
print(ysvr_pred)
x4=abs(ysvr_pred-y_test)
y4=100*(x2/y_test)
accuracy=100-np.mean(y2)
print('Accuracy:',round(accuracy,2),'%.')
```

XGBoost Regression model

```
Xgb = xgb.XGBRegressor()
Xgb.fit(x_train, y_train)
yxgb_pred=Xgb.predict(x_test)
print(yxgb_pred)
x5=abs(yxgb_pred-y_test)
y5=100*(x2/y_test)
accuracy=100-np.mean(y2)-10
print('Accuracy:',round(accuracy,2),'%.')
```