Selecting the best model with Best hyperparameters

In [1]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# train test split the data
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# import regression algorithms
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

#import grid search cv for cross validation
from sklearn.model_selection import GridSearchCV

# import preprocessors
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

In [2]:
```
# load dataset
df = sns.load_dataset('tips')


df.head()
```

Out[2]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [3]:

```
df.columns
```

Out[3]:

```
Index(['total_bill', 'tip', 'sex', 'smoker', 'day', 'time', 'size'], dtype='object')
```

Rergression Tasks

In [4]:

```
# select features and variables
X = df.drop('tip', axis=1)
y = df['tip']

# label encode categorical variables
le = LabelEncoder()
X['sex'] = le.fit_transform(X['sex'])
X['smoker'] = le.fit_transform(X['smoker'])
X['day'] = le.fit_transform(X['day'])
X['time'] = le.fit_transform(X['time'])
```

In [5]:

```
%%time
# split the data into train and test data with 80% training dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create a dictionaries of list of models to evaluate performance
models = {
          'LinearRegression' : LinearRegression(),
          'SVR' : SVR(),
          'DecisionTreeRegressor' : DecisionTreeRegressor(),
          'RandomForestRegressor' : RandomForestRegressor(),
          'KNeighborsRegressor' : KNeighborsRegressor(),
```

```
          'GradientBoostingRegressor' : GradientBoostingRegressor(),
          'XGBRegressor' : XGBRegressor()
          }

# train and predict each model with evaluation metrics as well making a for loop to iterate over the models

model_scores = []
for name, model in models.items():
    # fit each model from models on training data
    model.fit(X_train, y_train)

    # make prediction from each model
    y_pred = model.predict(X_test)
    metric = mean_absolute_error(y_test, y_pred)
    model_scores.append((name, metric))

    # print the performing metric
    print(name, 'MSE: ', mean_squared_error(y_test, y_pred))
    print(name, 'R2: ', r2_score(y_test, y_pred))
    print(name, 'MAE: ', mean_absolute_error(y_test, y_pred))
    print('\n')
# selecting the best model from all above models with evaluation metrics sorting method
sorted_models = sorted(model_scores, key=lambda x: x[1], reverse=False)
for model in sorted_models:
    print('Mean Absolute error for', f"{model[0]} is {model[1]: .2f}")
```

```
LinearRegression MSE:  0.6948129686287711
LinearRegression R2:  0.4441368826121931
LinearRegression MAE:  0.6703807496461158


SVR MSE:  0.538321847289585
SVR R2:  0.5693326496439823
SVR MAE:  0.5707097371316318


DecisionTreeRegressor MSE:  1.4143591836734695
DecisionTreeRegressor R2:  -0.13151328550238817
DecisionTreeRegressor MAE:  0.9285714285714286


RandomForestRegressor MSE:  0.9688009771428582
RandomForestRegressor R2:  0.22494145101268714
RandomForestRegressor MAE:  0.7729306122448979


KNeighborsRegressor MSE:  0.8382265306122448
KNeighborsRegressor R2:  0.3294034029001649
KNeighborsRegressor MAE:  0.7262448979591837


GradientBoostingRegressor MSE:  0.802513676733518
GradientBoostingRegressor R2:  0.3579743409570949
GradientBoostingRegressor MAE:  0.7260263944507737


XGBRegressor MSE:  0.7389215578875857
XGBRegressor R2:  0.40884920227805865
XGBRegressor MAE:  0.6721697168934103


Mean Absolute error for SVR is  0.57
Mean Absolute error for LinearRegression is  0.67
Mean Absolute error for XGBRegressor is  0.67
Mean Absolute error for GradientBoostingRegressor is  0.73
Mean Absolute error for KNeighborsRegressor is  0.73
Mean Absolute error for RandomForestRegressor is  0.77
Mean Absolute error for DecisionTreeRegressor is  0.93
```

```
CPU times: total: 1.2 s
Wall time: 294 ms
```

In [6]:

```python
%%time
# split the data into train and test data with 80% training dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create a dictionaries of list of models to evaluate performance
models = {
          'LinearRegression' : LinearRegression(),
          'SVR' : SVR(),
          'DecisionTreeRegressor' : DecisionTreeRegressor(),
          'RandomForestRegressor' : RandomForestRegressor(),
          'KNeighborsRegressor' : KNeighborsRegressor(),
          'GradientBoostingRegressor' : GradientBoostingRegressor(),
          'XGBRegressor' : XGBRegressor()
          }

# train and predict each model with evaluation metrics as well making a for loop to iterate over the models

model_scores = []
for name, model in models.items():
    # fit each model from models on training data
    model.fit(X_train, y_train)

    # make prediction from each model
    y_pred = model.predict(X_test)
    metric = r2_score(y_test, y_pred)
    model_scores.append((name, metric))

     # print the performing metric
    print(name, 'MSE: ', mean_squared_error(y_test, y_pred))
    print(name, 'R2: ', r2_score(y_test, y_pred))
    print(name, 'MAE: ', mean_absolute_error(y_test, y_pred))
    print('\n')
# selecting the best model from all above models with evaluation metrics sorting method
sorted_models = sorted(model_scores, key=lambda x: x[1], reverse=True)
for model in sorted_models:
    print('R_squared Score', f"{model[0]} is {model[1]: .2f}")
```

```
LinearRegression MSE:  0.6948129686287711
LinearRegression R2:  0.4441368826121931
LinearRegression MAE:  0.6703807496461158


SVR MSE:  0.538321847289585
SVR R2:  0.5693326496439823
SVR MAE:  0.5707097371316318


DecisionTreeRegressor MSE:  1.121761224489796
DecisionTreeRegressor R2:  0.10257044792896874
DecisionTreeRegressor MAE:  0.8189795918367349


RandomForestRegressor MSE:  1.010692044897961
RandomForestRegressor R2:  0.1914278285496377
RandomForestRegressor MAE:  0.7853591836734698


KNeighborsRegressor MSE:  0.8382265306122448
KNeighborsRegressor R2:  0.3294034029001649
KNeighborsRegressor MAE:  0.7262448979591837


GradientBoostingRegressor MSE:  0.7936081180416095
GradientBoostingRegressor R2:  0.3650989512336329
GradientBoostingRegressor MAE:  0.7230079530178968


XGBRegressor MSE:  0.7389215578875857
XGBRegressor R2:  0.40884920227805865
XGBRegressor MAE:  0.6721697168934103


R_squared Score SVR is  0.57
R_squared Score LinearRegression is  0.44
R_squared Score XGBRegressor is  0.41
R_squared Score GradientBoostingRegressor is  0.37
R_squared Score KNeighborsRegressor is  0.33
R_squared Score RandomForestRegressor is  0.19
R_squared Score DecisionTreeRegressor is  0.10
```

```
CPU times: total: 1.44 s
Wall time: 323 ms
```

In [7]:

```
%%time
# split the data into train and test data with 80% training dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create a dictionaries of list of models to evaluate performance
models = {
          'LinearRegression' : LinearRegression(),
          'SVR' : SVR(),
          'DecisionTreeRegressor' : DecisionTreeRegressor(),
          'RandomForestRegressor' : RandomForestRegressor(),
          'KNeighborsRegressor' : KNeighborsRegressor(),
          'GradientBoostingRegressor' : GradientBoostingRegressor(),
          'XGBRegressor' : XGBRegressor()
          }

# train and predict each model with evaluation metrics as well making a for loop to iterate over the models

model_scores = []
for name, model in models.items():
    # fit each model from models on training data
    model.fit(X_train, y_train)

    # make prediction from each model
    y_pred = model.predict(X_test)
    metric = mean_squared_error(y_test, y_pred)
    model_scores.append((name, metric))

    # # print the performing metric
    print(name, 'MSE: ', mean_squared_error(y_test, y_pred))
    print(name, 'R2: ', r2_score(y_test, y_pred))
    print(name, 'MAE: ', mean_absolute_error(y_test, y_pred))
    print('\n')
# selecting the best model from all above models with evaluation metrics sorting method
sorted_models = sorted(model_scores, key=lambda x: x[1], reverse=False)
for model in sorted_models:
    print('Mean Squared error for', f"{model[0]} is {model[1]: .2f}")
```

```
LinearRegression MSE:  0.6948129686287711
LinearRegression R2:  0.4441368826121931
LinearRegression MAE:  0.6703807496461158


SVR MSE:  0.538321847289585
SVR R2:  0.5693326496439823
SVR MAE:  0.5707097371316318


DecisionTreeRegressor MSE:  1.314895918367347
DecisionTreeRegressor R2:  -0.051940849156325575
DecisionTreeRegressor MAE:  0.9083673469387756


RandomForestRegressor MSE:  0.9624049597959197
RandomForestRegressor R2:  0.23005838218965147
RandomForestRegressor MAE:  0.7890673469387758


KNeighborsRegressor MSE:  0.8382265306122448
KNeighborsRegressor R2:  0.3294034029001649
KNeighborsRegressor MAE:  0.7262448979591837


GradientBoostingRegressor MSE:  0.8121705167201488
GradientBoostingRegressor R2:  0.3502486918666966
GradientBoostingRegressor MAE:  0.7303274108629866


XGBRegressor MSE:  0.7389215578875857
XGBRegressor R2:  0.40884920227805865
XGBRegressor MAE:  0.6721697168934103


Mean Squared error for SVR is  0.54
Mean Squared error for LinearRegression is  0.69
Mean Squared error for XGBRegressor is  0.74
Mean Squared error for GradientBoostingRegressor is  0.81
Mean Squared error for KNeighborsRegressor is  0.84
Mean Squared error for RandomForestRegressor is  0.96
Mean Squared error for DecisionTreeRegressor is  1.31
```

```
CPU times: total: 1.56 s
Wall time: 332 ms
```

Hyperparameter tuning:

In [8]:
```python
%%time
# Create a dictionaries of list of models to evaluate performance with hyperparameters
models = {
          'LinearRegression' : (LinearRegression(), {}),
          'SVR' : (SVR(), {'kernel': ['rbf', 'poly', 'sigmoid']}),
          'DecisionTreeRegressor' : (DecisionTreeRegressor(), {'max_depth': [None, 5, 10]}),
          'RandomForestRegressor' : (RandomForestRegressor(), {'n_estimators': [10, 100]}),
          'KNeighborsRegressor' : (KNeighborsRegressor(), {'n_neighbors': np.arange(3, 100, 2)}),
          'GradientBoostingRegressor' : (GradientBoostingRegressor(), {'n_estimators': [10, 100]}),
          'XGBRegressor' : (XGBRegressor(), {'n_estimators': [10, 100]}),
          }

# train and predict each model with evaluation metrics as well making a for loop to iterate over the models

for name, (model, params) in models.items():
    # create a pipline
    pipeline = GridSearchCV(model, params, cv=5)

    # fit the pipeline
    pipeline.fit(X_train, y_train)

    # make prediction from each model
    y_pred = pipeline.predict(X_test)


    # print the performing metric
    print(name, 'MSE: ', mean_squared_error(y_test, y_pred))
    print(name, 'R2: ', r2_score(y_test, y_pred))
    print(name, 'MAE: ', mean_absolute_error(y_test, y_pred))
    print('\n')
```

```
LinearRegression MSE:  0.6948129686287711
LinearRegression R2:  0.4441368826121931
LinearRegression MAE:  0.6703807496461158


SVR MSE:  1.460718141299992
SVR R2:  -0.1686013018011976
SVR MAE:  0.8935334948775431


DecisionTreeRegressor MSE:  0.8774153020453993
DecisionTreeRegressor R2:  0.298051667053291
DecisionTreeRegressor MAE:  0.7189481629481629


RandomForestRegressor MSE:  0.9783969628571446
RandomForestRegressor R2:  0.2172644864561979
RandomForestRegressor MAE:  0.7812693877551026


KNeighborsRegressor MSE:  0.6640950568462677
KNeighborsRegressor R2:  0.4687117753876745
KNeighborsRegressor MAE:  0.6203721488595437


GradientBoostingRegressor MSE:  0.8106801524004932
GradientBoostingRegressor R2:  0.35144101065487676
GradientBoostingRegressor MAE:  0.7657809818712309


XGBRegressor MSE:  0.6624107100882575
XGBRegressor R2:  0.4700592836840687
XGBRegressor MAE:  0.6549163442728472


CPU times: total: 48.7 s
Wall time: 15.2 s
```

In [ ]:

```
# Create a dictionaries of list of models to evaluate performance with hyperparameters
models = {
          'LinearRegression' : (LinearRegression(), {}),
          'SVR' : (SVR(), {'kernel': ['rbf', 'poly', 'sigmoid'], 'C': [0.1, 1, 10], 'gamma': [1, 0.1, 0.01], 'epsilon
```

```
          'DecisionTreeRegressor' : (DecisionTreeRegressor(), {'max_depth': [None, 5, 10], 'splitter': ['best', 'rand
          'RandomForestRegressor' : (RandomForestRegressor(), {'n_estimators': [10, 100, 1000], 'max_depth': [None, 5
          'KNeighborsRegressor' : (KNeighborsRegressor(), {'n_neighbors': np.arange(3, 100, 2), 'weights': ['uniform
          'GradientBoostingRegressor' : (GradientBoostingRegressor(), {'loss': ['ls', 'lad', 'huber', 'quantile'], 'n
          'XGBRegressor' : (XGBRegressor(), {'n_estimators': [10, 100, 1000], 'learning_rate': [0.1, 0.01, 0.001]}),
          }

# train and predict each model with evaluation metrics as well making a for loop to iterate over the models

for name, (model, params) in models.items():
    # create a pipline
    pipeline = GridSearchCV(model, params, cv=5)

    # fit the pipeline
    pipeline.fit(X_train, y_train)

    # make prediction from each model
    y_pred = pipeline.predict(X_test)


    # print the performing metric
    print(name, 'MSE: ', mean_squared_error(y_test, y_pred))
    print(name, 'R2: ', r2_score(y_test, y_pred))
    print(name, 'MAE: ', mean_absolute_error(y_test, y_pred))
    print('\n')
```

```
LinearRegression MSE:  0.6948129686287711
LinearRegression R2:  0.4441368826121931
LinearRegression MAE:  0.6703807496461158
```

In [ ]:

Add preprocessor inside the pipeline

Create a dictionaries of list of models to evaluate performance with hyperparameters models = { 'LinearRegression' : (LinearRegression(), {}), 'SVR' : (SVR(), {'kernel': ['rbf', 'poly', 'sigmoid'], 'C': [0.1, 1, 10], 'gamma': [1, 0.1, 0.01], 'epsilon': [0.1, 0.01, 0.001]}), 'DecisionTreeRegressor' : (DecisionTreeRegressor(), {'max_depth': [None, 5, 10], 'splitter': ['best', 'random']}), 'RandomForestRegressor' : (RandomForestRegressor(), {'n_estimators': [10, 100, 1000], 'max_depth': [None, 5, 10]}), 'KNeighborsRegressor' : (KNeighborsRegressor(), {'n_neighbors': np.arange(3, 100, 2), 'weights': ['uniform', 'distance']}), 'GradientBoostingRegressor' : (GradientBoostingRegressor(), {'loss': ['ls', 'lad', 'huber', 'quantile'], 'n_estimators': [10, 100, 1000]}), 'XGBRegressor' : (XGBRegressor(), {'n_estimators': [10, 100, 1000], 'learning_rate': [0.1, 0.01, 0.001]}), } # train and

predict each model with evaluation metrics as well making a for loop to iterate over the models for name, (model, params) in models.items(): # create a pipline with preprocessor pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)]) # make a grid search cv to tune the hyperparameter grid_search = GridSearchCV(pipeline, params, cv=5) # fit the pipeline grid_search.fit(X_train, y_train) # make prediction from each model y_pred = grid_search.predict(X_test) # print the performing metric print(name, 'MSE: ', mean_squared_error(y_test, y_pred)) print(name, 'R2: ', r2_score(y_test, y_pred)) print(name, 'MAE: ', mean_absolute_error(y_test, y_pred)) print('\n')

Classifiers:

In [ ]:
```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

# dont show warnings
import warnings
warnings.filterwarnings('ignore')

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Create a dictionary of classifiers to evaluate
classifiers = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(),
    'KNN': KNeighborsClassifier()
}

# Perform k-fold cross-validation and calculate the mean accuracy
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

for name, classifier in classifiers.items():
    scores = cross_val_score(classifier, X, y, cv=kfold)
    accuracy = np.mean(scores)
```

```python
    print("Classifier:", name)
    print("Mean Accuracy:", accuracy)
    print()
```

In [ ]:

```python

```