Starter point for the Machine Learning element of the coursework

This part of the coursework uses the same dataset as used in the first part of the coursework. In this part of the coursework you are going to develop three machine learning models for predicting the **'median_house_value'** from the dataset. The focus here is on comparing the different models and looking at how you can improve them. You do not need to use a cleaned up version of the data (i.e. perform outlier removal in advance) but you may if you wish although please comment on this.

Everything needed to complete this assignment should be available in the course slides, but external information from the interenet may prove useful and is encouraged. Please provide citations for resources used in the course.

IMPORT PACKAGES

```
!pip install numpy
!pip install pandas
!pip install matplotlib
!pip install scikit-learn
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.23.5)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.3.post1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.23.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.46.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotli
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.
```

Load libraries and read in data

```
import pandas as pd
houses = pd.read_csv('https://raw.githubusercontent.com/PaoloMissier/CSC3831-2021-22/main/IMPUTATION/TARGET-DATASETS/ORIGINA

houses.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   median_house_value  20640 non-null  float64
 1   median_income       20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20640 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   latitude            20640 non-null  float64
 8   longitude           20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

Data Preparation

In this section you will perform feature selection, feature normalisation, and provide a rationale for your actions.

Feature Selection

Think about which features may be useful in predicting **'median_house_value'**, are all features in the provided data set useful? Is a subset all that's needed? What techniques can you utilise to make this determination?

The features population, total rooms, median income, and households are considered valuable indicators for predicting the median house value. Population reflects the demand for housing, while total rooms captures the size of the property. Median income represents the purchasing power of individuals, and households provide insights into housing supply and demand dynamics. Together, these features include economic and spatial aspects which can be strongly impact the median house value.

Feature Normalisation

Think about what normalisation/standardisation methods you should apply to the dataset given what you understand about the raw data.

I've select four features:
1) Rooms per Household - Houses with more rooms usually mean the house is larger. Generally, larger houses tend to have higher values, and by calculating this, we can capture the housing density or the availability of space, which can be indicative of the housing market value.
2) People per Household - This can reflect the population density or the occupation level of housing in different areas, and can also be an indicator of housing affordability, more number of people per house indicates for requirement for larger houses which can affect the median house value quite directly.
3) Occupancy per Room - This feature further examines the occupancy level of each house, can give insights about overcrowding/underutilization of space. Higher occupancy per room indicates the demand for larger houses or the scarcity of housing might be higher, impacting the median house value.
4) Income per household - Higher-income households generally have a higher purchasing power and can afford more expensive houses. This indicates the affordability aspect and its influence on the median house value.

```
# Calculate normalized features

houses_normalised = houses.copy()

houses_normalised['rooms_per_household'] = houses['total_rooms'] / houses['households']
houses_normalised['people_per_household'] = houses['population'] / houses['households']
houses_normalised['occupancy_per_room'] = houses['population'] / houses['total_rooms']
houses_normalised['income_per_household'] = houses['median_income'] / houses['households']


# Display the new features
print(houses_normalised[['rooms_per_household', 'people_per_household','occupancy_per_room', 'income_per_household']].head()
```

```
   rooms_per_household  people_per_household  occupancy_per_room  \
0             6.984127              2.555556            0.365909
1             6.238137              2.109842            0.338217
2             8.288136              2.802260            0.338105
3             5.817352              2.547945            0.437991
4             6.281853              2.181467            0.347265

   income_per_household
0              0.066073
1              0.007295
2              0.041002
3              0.025768
4              0.014850
```

Rationale

Provide your rationale for both Feature Selection and Feature Normalisation here.

Feature selection is important to identify the most relevant features, simplify the model, and reduce overfitting. Feature normalization ensures that features are on a similar scale, improves algorithm performance, and enhances interpretability of the data.

Train, Validate, Test Split

In this section you will perform a train, validate, test split utilisation the knowledge learned in class and provide a rationale for your actions.

Functions of Calculate Mean-Sqaured Error, Mean-Absolute Error and R-Squared Error

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
import numpy as np

def calc_mse(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    return mse

def calc_mae(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    return mae

def calc_r2(model, X_test, y_test):
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    return r2
```

Function to Print Evaluation of the performance of Models and Returns MSE.

```
def evaluate(model, model_name, X_test, y_test):
    mse = calc_mse(model, X_test, y_test)
    mae = calc_mae(model, X_test, y_test)
    r2 = calc_r2(model, X_test, y_test)

    print(f'{model_name} Performance:')
    print("Mean Squared Error (MSE):", mse)
    print("Mean Absolute Error (MAE):", mae)
    print("R-squared:", r2)

    return mse, mae, r2
```

Function to Compare the three scores

```
def compare_mse(mse1, mse2):

    diff_mse = 100 * (mse2 - mse1) / mse1

    if diff_mse >= 0:
        print('Increase in Mean Squared Error by {:0.2f}%'.format(np.abs(diff_mse)))
    else:
        print('Decrease in Mean Squared Error by {:0.2f}%'.format(np.abs(diff_mse)))

def compare_mae(mae1, mae2):

    diff_mae = 100 * (mae2 - mae1) / mae1

    if diff_mae >= 0:
        print('Increase in Mean Absolute Error by {:0.2f}%'.format(np.abs(diff_mae)))
    else:
        print('Decrease in Mean Absolute Error by {:0.2f}%'.format(np.abs(diff_mae)))

def compare_r2(r2_1, r2_2):

    diff_r2 = 100 * (r2_2 - r2_1) / r2_1

    if diff_r2 >= 0:
        print('Increase in R-Sqaured by {:0.2f}%'.format(np.abs(diff_r2)))
    else:
        print('Decrease in R-Sqaured by {:0.2f}%'.format(np.abs(diff_r2)))
```

Function to Compare The Different Models

```
def compare_models(model1, model2, model_name1, model_name2, X_test, y_test):
    mse_model_1 = calc_mse(model1, X_test, y_test)
    mse_model_2 = calc_mse(model2, X_test, y_test)

    mae_model_1 = calc_mae(model1, X_test, y_test)
    mae_model_2 = calc_mae(model2, X_test, y_test)

    r2_model_1 = calc_r2(model1, X_test, y_test)
    r2_model_2 = calc_r2(model2, X_test, y_test)

    diff_mse = 100 * (mse_model_2 - mse_model_1) / mse_model_1
    diff_mae = 100 * (mae_model_2 - mae_model_1) / mae_model_1
    diff_r2 = 100 * (r2_model_2 - r2_model_1) / r2_model_1

    print('{} gives {:0.2f}% better Mean Squared Error Score than {}'.format(model_name2, diff_mse, model_name1))
    print('{} gives {:0.2f}% better Mean Absolute Error Score than {}'.format(model_name2, diff_mae, model_name1))
    print('{} gives {:0.2f}% better R-Sqaured than {}\n'.format(model_name2, diff_r2, model_name1))
```

Perform Data Split

Utilise the information from class or online to split your data into train, validate, and test partitions.

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Scaling
scaler = StandardScaler()
houses_scaled = scaler.fit_transform(houses)
houses_scaled = pd.DataFrame(houses_scaled, columns=houses.columns)

# Spliting Data (80% Training and 20% Testing)
X = houses_scaled.drop(columns=['median_house_value'])
y = houses_scaled['median_house_value']

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

X_train_scaled = scaler.fit_transform(X_train)

# Further split the train set into train and validation sets (80% train, 20% validation)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

Rationale

Provide a rationale for how you made your train, test, split decision.

I chose an 80-20 split for the train and test sets, which is a commonly used ratio in machine learning projects. With 80% of the data allocated for training, the model can learn from a substantial amount of examples, while the remaining 20% serves as a reliable test set to evaluate its performance on unseen data.

Metric Selection

In this section your will make appropriate metric selection for analysing your models and provide a rationale.

Select Appropriate Metrics

Think about the models you are building, and what the appropriate metrics and scoring should be for those models.

For this tasks, I will use common evaluation metrics including mean squared error (MSE), mean absolute error (MAE), and R-squared (coefficient of determination).

Rationale

Provide a rationale for your metrics and scoring selection.

I selected evaluation metrics (MSE, MAE, R-squared) to assess the regression model's accuracy, error magnitude, and goodness-of-fit. These metrics provide insights into the model's predictive capabilities, aiding informed decisions for predicting house values.

Machine Learning Model 1

In this section your will select an appropriate machine learning model for predicting '**median_house_value**', apply it to the dataset to perform this prediction on the test set created in the Train, Validate, Test Split section, and comment on the predictive ability of the model you selected.

Select and Build a Machine Learning Model 1

Think about the task at hand, and select an appropriate model to build on the train and validate data. Try different sets of hyper-parameters to improve your model.

RANDOM FOREST

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

# Random Forest Modelling
rf_model = RandomForestRegressor(n_estimators=100, max_depth=10)
rf_model.fit(X_train, y_train)

# Evaluating the Base Model
base_rf_model = evaluate(rf_model, 'Random Forest Base Model', X_test, y_test)
```

```
Random Forest Base Model Performance:
Mean Squared Error (MSE): 0.2245331280732427
Mean Absolute Error (MAE): 0.31589171910306724
R-squared: 0.7718443943618413
```

First Set of Hyper-Parameter: Randomised Search CV

```
from sklearn.model_selection import RandomizedSearchCV
from math import sqrt

# Base Random Forest model
base_model_cv_rf = RandomForestRegressor(random_state=42)
base_model_cv_rf.fit(X_train, y_train)

# Hyperparameter grid for Randomized Search CV
param_grid_cv_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt']
}

# Randomized Search CV
random_search_cv_rf = RandomizedSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_distributions=param_grid_cv_rf,
    n_iter=4,
    scoring='neg_mean_squared_error',
    cv=5,
    random_state=42
)

# Fit Randomized Search CV to training data
random_search_cv_rf.fit(X_train, y_train)

# Best parameter Search
best_search_cv_rf = random_search_cv_rf.best_estimator_

# Model's performance
base_rf_cv_mse, base_rf_cv_mae, base_rf_cv_r2 = evaluate(base_model_cv_rf, 'Random Forest Base Model', X_test, y_test)
print("\n")
best_rf_cv_mse, best_rf_cv_mae, best_rf_cv_r2 = evaluate(best_search_cv_rf, 'Random Forest Best Search Model', X_test, y_tes
print("\n")
compare_mse(base_rf_cv_mse, best_rf_cv_mse)
compare_mae(base_rf_cv_mae, best_rf_cv_mae)
compare_r2(base_rf_cv_r2, best_rf_cv_r2)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
```

```
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been d
  warn(
Random Forest Base Model Performance:
Mean Squared Error (MSE): 0.19469252247612864
Mean Absolute Error (MAE): 0.2857740037189359
R-squared: 0.8021664296937416


Random Forest Best Search Model Performance:
Mean Squared Error (MSE): 0.22561546069471877
Mean Absolute Error (MAE): 0.31593790140489986
R-squared: 0.7707446000603329


Increase in Mean Squared Error by 15.88%
Increase in Mean Absolute Error by 10.56%
Decrease in R-Sqaured by 3.92%
```

Second Set of Hyper-Parameter: Randomised Search Cross-Validation

```
# Base Random Forest model
base_model_scv_rf = RandomForestRegressor()
base_model_scv_rf.fit(X_train, y_train)

n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
max_features = ['log2', 'sqrt', None]
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]

random_grid_rf = {
    'n_estimators': n_estimators,
    'max_features': max_features,
    'max_depth': max_depth,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf,
    'bootstrap': bootstrap
}

# Cross-Validation
scv_search_rf = RandomizedSearchCV(estimator=base_model_scv_rf,
                                   param_distributions=random_grid_rf,
                                   n_iter=30,
                                   cv=3,
                                   verbose=2,
                                   random_state=42,
                                   n_jobs=-1)

# Fit Randomized Cross-Validation to training data
scv_search_rf.fit(X_train, y_train)

# Best parameter Search
best_search_scv_rf = scv_search_rf.best_estimator_

# Model's performance
base_rf_scv_mse, base_rf_scv_mae, base_rf_scv_r2 = evaluate(base_model_scv_rf, 'Random Forest Base Model', X_test, y_test)
print("\n")
best_rf_scv_mse, best_rf_scv_mae, best_rf_scv_r2 = evaluate(best_search_scv_rf, 'Random Forest Best Search Model', X_test, y
print("\n")
compare_mse(base_rf_scv_mse, best_rf_scv_mse)
compare_mae(base_rf_scv_mae, best_rf_scv_mae)
compare_r2(base_rf_scv_r2, best_rf_scv_r2)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker stopped whi
  warnings.warn(
Random Forest Base Model Performance:
Mean Squared Error (MSE): 0.19604711128496904
Mean Absolute Error (MAE): 0.286184944366063
R-squared: 0.8007899867932052


Random Forest Best Search Model Performance:
Mean Squared Error (MSE): 0.18942617554365418
Mean Absolute Error (MAE): 0.2851348753127427
R-squared: 0.8075177405857693


Decrease in Mean Squared Error by 3.38%
Decrease in Mean Absolute Error by 0.37%
Increase in R-Sqaured by 0.84%
```

Use Model 1 to Predict on Test Data

Use the model you've trained to predict '**median_house_value**' on the test data.

```
# Predict 'median_house_value' using the trained model on the test data
y_pred_best_model_cv_rf = best_search_cv_rf.predict(X_test)
comparison_df_best_model_cv_rf = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_best_model_cv_rf})

# First few instances for comparison
print(comparison_df_best_model_cv_rf.head())

# Predict 'median_house_value' using the trained model on the test data
y_pred_best_model_scv_rf = best_search_scv_rf.predict(X_test)
comparison_df_best_model_scv_rf = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_best_model_scv_rf})

# First few instances for comparison
print(comparison_df_best_model_scv_rf.head())
```

```
          Actual  Predicted
20046 -1.379252  -1.315465
3024  -1.395718  -1.143379
15663  2.540411   2.322297
20484  0.101776   0.526281
9814   0.616539   0.447497
          Actual  Predicted
20046 -1.379252  -1.347455
3024  -1.395718  -1.009125
15663  2.540411   2.303932
20484  0.101776   0.492772
9814   0.616539   0.418237
```

Rationale

Random Forest is a suitable choice for this task because it can predict median house values very well as it can handle both numerical and categorical features, and handle outlier very well(which we saw right above), it also provides quite reliable estimates. However, its not been very helpful here but usually, it does play an impactful role in model training and testing.

Double-click (or enter) to edit

Comment on Predictive Ability

Think about the metrics and scoring received from the training and testing components. Think about the generalisability and quality of your results.

For the predictive ability of the optimized RF Model, it can be seen that a there is a strong relation between the predicted and the actual median house values. The metrics derived from both training and testing phases showcase that the model is adept in identifying complex relationships within the dataset. A good R^2 value suggests that major proportion of variance in house values have been accounted for by the model, therefore giving reliable predictions.

The model's generalizability is shown by the findings, which demonstrate that it has discovered underlying patterns in the training data that apply to the unknown test set. This feature indicates that the model is well-tuned and balances fitting the training data with retaining flexibility to adjust to new data. Because of the model's performance on the test data, one can be confident on applying this model in real-world situations where it can provide insightful information and help to attain the housing values.

Machine Learning Model 2

In this section your will select an appropriate machine learning model for predicting '**median_house_value**', apply it to the dataset to perform this prediction on the test set created in the Train, Validate, Test Split section, and comment on the predictive ability of the model you selected.

Select and Build a Machine Learning Model 2

Think about the task at hand, and select an appropriate model to build on the train and validate data. Try different sets of hyper-parameters to improve your model.

```
# Support Vector Model
from sklearn.svm import SVR

# Creating and training the model
svr = SVR()
svr.fit(X_train, y_train)
base_model_svr = evaluate(svr, 'Support Vector base Model', X_test, y_test)
```

First Set of Hyper-Parameter: Randomised Search CV

```
from sklearn.pipeline import Pipeline
from scipy.stats import uniform, reciprocal
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RandomizedSearchCV

# Pipeline with preprocessing and SVR model
pipeline_svr = Pipeline([
    ('scaler', StandardScaler()),
    ('svr', SVR())
])

# Parameter grid for RandomizedSearchCV
param_grid_svr = {
    'svr__kernel': ['linear', 'rbf'],
    'svr__C': reciprocal(1, 100),
    'svr__gamma': reciprocal(0.01, 1)
}

# Create the RandomizedSearchCV object
random_search_svr = RandomizedSearchCV(estimator=pipeline_svr,
                                       param_distributions=param_grid_svr,
                                       n_iter=4,
                                       scoring='neg_mean_squared_error',
                                       cv=3,
                                       random_state=42)

# Fit the RandomizedSearchCV object to the data
random_search_svr.fit(X_train, y_train)

# Best parameter Search
best_search_cv_svr = random_search_svr.best_estimator_

# Model's performance
base_svr_cv_mse, base_svr_cv_mae, base_svr_cv_r2 = evaluate(svr, 'Support Vector with Randomized Search CV Base Model', X_te
print("\n")
best_svr_cv_mse, best_svr_cv_mae, best_svr_cv_r2 = evaluate(best_search_cv_svr, 'Support Vector Best Grid Search Model', X_t
print("\n")
compare_mse(base_svr_cv_mse, best_svr_cv_mse)
compare_mae(base_svr_cv_mae, best_svr_cv_mae)
compare_r2(base_svr_cv_r2, best_svr_cv_r2)
```

```
Support Vector with Randomized Search CV Base Model Performance:
Mean Squared Error (MSE): 0.25150612259191424
Mean Absolute Error (MAE): 0.33433607829257045
R-squared: 0.7444362343584996


Support Vector Best Grid Search Model Performance:
Mean Squared Error (MSE): 0.22882370035404007
Mean Absolute Error (MAE): 0.31366066919910673
R-squared: 0.7674846006616426


Decrease in Mean Squared Error by 9.02%
Decrease in Mean Absolute Error by 6.18%
Increase in R-Sqaured by 3.10%
```

Second Set of Hyper-Parameter: Randomised Search Cross-Validation

```
base_model_svr = SVR()

# Defining Parameter Grid for Support Vector
kernel = ['linear', 'rbf', 'sigmoid']
gamma = ['scale', 'auto']
epsilon = np.arange(start=0.1, stop=0.6, step=0.1)
shrinking = [True, False]
cache_size = [int(x) for x in np.linspace(start=200, stop=1000, num=9)]

random_grid_svr = {
    'kernel': kernel,
    'gamma': gamma,
    'epsilon': epsilon,
    'shrinking': shrinking,
    'cache_size': cache_size
}


# Cross-Validation
svr_search_scv_svr = RandomizedSearchCV(estimator=base_model_svr,
                                param_distributions=random_grid_svr,
                                n_iter=30,
                                cv=3,
                                verbose=2,
                                random_state=42,
                                n_jobs=-1)

# Fit Randomized Cross-Validation to training data
svr_search_scv_svr.fit(X_train, y_train)


# Best parameter Search
best_search_svr_scv = svr_search_scv_svr.best_estimator_

# Model's performance
base_svr_scv_mse, base_svr_scv_mae, base_svr_scv_r2 = evaluate(svr, 'Support Vector Base Model', X_test, y_test)
print("\n")
best_svr_scv_mse, best_svr_scv_mae, best_svr_scv_r2 = evaluate(best_search_svr_scv, 'Support Vector Best Search Model', X_te
print("\n")
compare_mse(base_svr_scv_mse, best_svr_scv_mse)
compare_mae(base_svr_scv_mae, best_svr_scv_mae)
compare_r2(base_svr_scv_r2, best_svr_scv_r2)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
Support Vector Base Model Performance:
Mean Squared Error (MSE): 0.25150612259191424
Mean Absolute Error (MAE): 0.33433607829257045
R-squared: 0.7444362343584996


Support Vector Best Search Model Performance:
Mean Squared Error (MSE): 0.2504465264264307
Mean Absolute Error (MAE): 0.33720693308217126
R-squared: 0.7455129253882034


Decrease in Mean Squared Error by 0.42%
Increase in Mean Absolute Error by 0.86%
Increase in R-Sqaured by 0.14%
```

Use Model 2 to Predict on Test Data

Use the model you've trained to predict '**median_house_value**' on the test data.

```
# Predict 'median_house_value' using the trained model on the test data
y_pred_best_model_cv_svr = best_search_svr_scv.predict(X_test)
comparison_df_best_model_cv_svr = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_best_model_cv_svr})

# First few instances for comparison
print(comparison_df_best_model_cv_svr.head())

# Predict 'median_house_value' using the trained model on the test data
y_pred_best_model_scv_svr = best_search_cv_svr.predict(X_test)
comparison_df_best_model_scv_svr = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_best_model_scv_svr})

# First few instances for comparison
print(comparison_df_best_model_scv_svr.head())
```

```
              Actual  Predicted
20046 -1.379252  -1.245011
3024  -1.395718  -0.393106
```

```
15663  2.540411   2.297324
20484  0.101776   0.305287
9814   0.616539   0.528013
          Actual  Predicted
20046 -1.379252  -1.370268
3024  -1.395718  -0.616170
15663  2.540411   3.041298
20484  0.101776   0.346181
9814   0.616539   0.718314
```

Rationale

Provide a rationale for model

SVR is a very relevant choice for predicting median house values as it can handle nonlinear relationships, outliers, and high-dimensional features very effectively. SVR helps to minimize the error and provide accurate predictions for the data and requirement.

Comment on Predictive Ability

Think about the metrics and scoring received from the training and testing components. Think about the generalisability and quality of your results.

The SVM model's relatively high R-squared score shows that it can display good proportion of variance in the housing value. However, the MSE and MAE suggest prediction accuracy can be improved. The hyperparameter tuning sessions via RandomizedSearchCV resulted in a notable decrease in MSE and MAE and an improvement in the R-squared value, showing the benefits of model optimization.

The dataset looks to suit the SVM model, which performs in higher-dimensional areas. Given the slight gains after intensive hyperparameter adjustment, the default settings may have solved this issue near-optimally. This shows the SVM's predictive performance may be limited by feature representation or model complexity.

The SVM's home value forecasts are accurate for broad estimates but may not be accurate for more complex ones. The SVM model is generalizable, but it needs more validation across datasets, maybe stretching regions or housing market situations.

Machine Learning Model 3

In this section your will select an appropriate machine learning model for predicting **'median_house_value'**, apply it to the dataset to perform this prediction on the test set created in the Train, Validate, Test Split section, and comment on the predictive ability of the model you selected.

Select and Build a Machine Learning Model 3

Think about the task at hand, and select an appropriate model to build on the train and validate data. Try different sets of hyper-parameters to improve your model.

eXtreme Gradient Boosting

```
# Gradient Boosting (XGBoost)
import xgboost as xgb

# Creating and training the XGBoost model
xgbr = xgb.XGBRegressor()
xgbr.fit(X_train, y_train)
```

```
▾                                XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

First Set of Hyper-Parameter: Randomised Search CV

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform, randint

# Defining the parameter grid for RandomizedSearchCV
param_grid_xgb_cv = {
    'n_estimators': randint(100, 1000),
    'learning_rate': uniform(0.01, 0.5),
    'max_depth': randint(3, 10),
    'subsample': uniform(0.6, 0.4),
    'colsample_bytree': uniform(0.6, 0.4),
    'gamma': uniform(0, 5)
}

# Creating the RandomizedSearchCV object
random_search_xgb_cv = RandomizedSearchCV(estimator=xgbr,
                                          param_distributions=param_grid_xgb_cv,
                                          n_iter=4,
                                          scoring='neg_mean_squared_error',
                                          cv=5,
                                          random_state=42,
                                          n_jobs=-1)


# Fitting the RandomizedSearchCV object to the data
random_search_xgb_cv.fit(X_train, y_train)

# Getting the best model
best_search_xgb_cv = random_search_xgb_cv.best_estimator_

# Model's performance
base_xgb_cv_mse, base_xgb_cv_mae, base_xgb_cv_r2 = evaluate(xgbr, 'XGBoost Base Model', X_test, y_test)
print("\n")
best_xgb_cv_mse, best_xgb_cv_mae, best_xgb_cv_r2 = evaluate(best_search_xgb_cv, 'XGBoost Best Search Model', X_test, y_test)
print("\n")

compare_mse(base_xgb_cv_mse, best_xgb_cv_mse)
compare_mae(base_xgb_cv_mae, best_xgb_cv_mae)
compare_r2(base_xgb_cv_r2, best_xgb_cv_r2)
```

```
XGBoost Base Model Performance:
Mean Squared Error (MSE): 0.1764602416667473
Mean Absolute Error (MAE): 0.2802490563802504
R-squared: 0.8206928587598012


XGBoost Best Search Model Performance:
Mean Squared Error (MSE): 0.192223789122506
Mean Absolute Error (MAE): 0.3010158368473712
R-squared: 0.8046749920528389


Increase in Mean Squared Error by 8.93%
Increase in Mean Absolute Error by 7.41%
Decrease in R-Sqaured by 1.95%
```

Second Set of Hyper-Parameter: Randomised Search Cross-Validation

```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform, randint

# Base model for XGBoost
base_model_xgb = xgb.XGBRegressor(objective='reg:squarederror')
base_model_xgb.fit(X_train, y_train)

# Defining Parameter Grid for XGBoost
param_grid_xgb_scv = {
    'n_estimators': randint(100, 1000),
    'max_depth': randint(3, 10),
    'learning_rate': uniform(0.01, 0.59),
    'subsample': uniform(0.5, 0.4),
    'colsample_bytree': uniform(0.5, 0.4),
    'gamma': uniform(0, 5)
}

# Randomized Search Cross-Validation
xgb_search_scv = RandomizedSearchCV(estimator=base_model_xgb,
                                    param_distributions=param_grid_xgb_scv,
                                    n_iter=30,
                                    cv=3,
                                    verbose=2,
                                    random_state=42,
                                    n_jobs=-1)

# Fit Randomized Search Cross-Validation to training data
xgb_search_scv.fit(X_train, y_train)

# Best parameter Search
best_search_xgb_scv = xgb_search_scv.best_estimator_

# Model's Performace
base_xgb_scv_mse, base_xgb_scv_mae, base_xgb_scv_r2 = evaluate(base_model_xgb, 'XGBoost Base Model', X_test, y_test)
print("\n")
best_xgb_scv_mse, best_xgb_scv_mae, best_xgb_scv_r2 = evaluate(best_search_xgb_scv, 'XGBoost Best Model', X_test, y_test)
print("\n")

compare_mse(base_xgb_scv_mse, best_xgb_scv_mse)
compare_mae(base_xgb_scv_mae, best_xgb_scv_mae)
compare_r2(base_xgb_scv_r2, best_xgb_scv_r2)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
XGBoost Base Model Performance:
Mean Squared Error (MSE): 0.1764602416667473
Mean Absolute Error (MAE): 0.2802490563802504
R-squared: 0.8206928587598012


XGBoost Best Model Performance:
Mean Squared Error (MSE): 0.16163999083328628
Mean Absolute Error (MAE): 0.26951718654014584
R-squared: 0.8357522102846003


Decrease in Mean Squared Error by 8.40%
Decrease in Mean Absolute Error by 3.83%
Increase in R-Sqaured by 1.83%
```

Use Model 3 to Predict on Test Data

Use the model you've trained to predict '**median_house_value**' on the test data.

```
# Predict 'median_house_value' using the trained model on the test data
y_pred_best_model_cv_xgb = best_search_xgb_cv.predict(X_test)
comparison_df_best_model_cv_xgb = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_best_model_cv_xgb})

# First few instances for comparison
print(comparison_df_best_model_cv_xgb.head())

# Predict 'median_house_value' using the trained model on the test data
y_pred_best_model_scv_xgb = best_search_xgb_scv.predict(X_test)
comparison_df_best_model_scv_xgb = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_best_model_scv_xgb})

# First few instances for comparison
print(comparison_df_best_model_scv_xgb.head())
```

```
              Actual  Predicted
20046 -1.379252  -1.480393
3024  -1.395718  -0.667132
15663  2.540411   2.707046
```

```
20484  0.101776   0.584431
9814   0.616539   0.100060
         Actual  Predicted
20046 -1.379252  -1.385794
3024  -1.395718  -1.013201
15663  2.540411   3.151627
20484  0.101776   0.473945
9814   0.616539   0.478283
```

Rationale

Provide a rationale for model

XGBoost is also a well-suited model for predicting median house values as it excels in capturing complex interactions between features, handles missing values pretty well, and it provides high predictive accuracy, as evident above. It can also handle large scale dataset much easily than others as it took less time to execute itself fully.

Comment on Predictive Ability

Think about the metrics and scoring received from the training and testing components. Think about the generalisability and quality of your results.

The XGBoost model's high R-squared value shows its ability to forecast house value variation. With the lowest MSE and MAE among evaluated models, the model predicts accurately and reliably.

The first set of hyperparameter tuning indicated a small decrease in predictive performance, showing the original values were effective. The second tuning session reduced MSE and MAE and increased R-squared. This improvement indicated the effect of tuning XGBoost's hyperparameter and possibility for more optimisation.

The XGBoost approach seems generalizable across datasets. However, the performance variation before and after hyperparameter adjustment shows the need for extensive validation to prevent overfitting and ensuring consistency. After adjusting, the model's metrics increased, suggesting that XGBoost can accurately predict house values with higher accuracy.

Comparison Between Models

In this section you will comment on the difference in predictive ability between models, the difference in analysis metrics between models, and the pros/cons of each model as you understand it.

```
# Compare best Random Forest model with the best SVR model
compare_models(best_search_scv_rf, best_search_cv_svr, 'Best Random Forest', 'Best SVR', X_test, y_test)

# Compare best SVR model with the best XGBoost model
compare_models(best_search_cv_svr, best_search_xgb_cv, 'Best SVR', 'Best XGBoost', X_test, y_test)

# Compare best XGBoost model with the best Random Forest model
compare_models(best_search_xgb_cv, best_search_scv_rf, 'Best XGBoost', 'Best Random Forest', X_test, y_test)

# You can also compare the base models with their respective best models
# Compare base Random Forest with its best model
compare_models(rf_model, best_search_scv_rf, 'Base Random Forest', 'Best Random Forest', X_test, y_test)

# Compare base SVR with its best model
compare_models(svr, best_search_cv_svr, 'Base SVR', 'Best SVR', X_test, y_test)

# Compare base XGBoost with its best model
compare_models(xgbr, best_search_xgb_cv, 'Base XGBoost', 'Best XGBoost', X_test, y_test)
```

```
Best SVR gives 20.80% better Mean Squared Error Score than Best Random Forest
Best SVR gives 10.00% better Mean Absolute Error Score than Best Random Forest
Best SVR gives -4.96% better R-Sqaured than Best Random Forest

Best XGBoost gives -15.99% better Mean Squared Error Score than Best SVR
Best XGBoost gives -4.03% better Mean Absolute Error Score than Best SVR
Best XGBoost gives 4.85% better R-Sqaured than Best SVR

Best Random Forest gives -1.46% better Mean Squared Error Score than Best XGBoost
Best Random Forest gives -5.28% better Mean Absolute Error Score than Best XGBoost
Best Random Forest gives 0.35% better R-Sqaured than Best XGBoost

Best Random Forest gives -15.64% better Mean Squared Error Score than Base Random Forest
Best Random Forest gives -9.74% better Mean Absolute Error Score than Base Random Forest
Best Random Forest gives 4.62% better R-Sqaured than Base Random Forest
```

```
Best SVR gives -9.02% better Mean Squared Error Score than Base SVR
Best SVR gives -6.18% better Mean Absolute Error Score than Base SVR
Best SVR gives 3.10% better R-Sqaured than Base SVR

Best XGBoost gives 8.93% better Mean Squared Error Score than Base XGBoost
Best XGBoost gives 7.41% better Mean Absolute Error Score than Base XGBoost
Best XGBoost gives -1.95% better R-Sqaured than Base XGBoost
```

Difference in Predictive Ability

Think about what you understand if the predictive ability from each model based on scores gained, comment on the difference between each model and suggest why you believe this to be the case. Additionally comment on the difference in hyper-parameter selection between your models.

I find variations in each model's prediction abilities due to algorithmic complexity and hyperparameter adjustment. For eg. the Random Forest model prevents overfitting by averaging many decision trees, offering good baseline prediction ability. Tuning the hyperparameters using Randomised Search CV improved performance, demonstrating that the model's original setup was not ideal and that it improved after going through a wider parameter space.

However, the Support Vector Machine model began with a lower R-squared value, suggesting a less precise data fit. I've read and I believe that the SVMs focus on margin maximisation and may not reflect the dataset's complexity without proper kernel and parameter selection. The SVM model enhanced predictive ability after tuning hyperparameters, showing the importance of kernel and regularisation strength selection for high-dimensional feature space.

XGBoost, at the start itself beat the other two model because of its gradient boosting architecture(that successively constructs trees to rectify faults and optimise efficiency). Interestingly, hyperparameter adjustment did not improve performance, showing that the default values were effective. XGBoost's regularised boosting strategy may not need substantial modification to attain great predictive accuracy while being amonst the fastest.

Hyperparameter selection differences amongst models show each algorithm's distinct qualities and sensitivities. Random Forest and SVM improved significantly with modification, suggesting their default settings were unsuitable for this dataset. In comparison, XGBoost's limited improvements post-tuning may indicate that its underlying algorithm is resistant to hyperparameter choice or because Randomised Search CV's search space does not contain the ideal set for actual performance improvements.

Difference in Analysis Metrics