Project 4: California Housing Price Prediction

MainCode

```
# Step1: Import all libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Step2: Load the data

# Step2.1: Read the “housing.csv” file from the folder into the program

housingData = pd.read_csv('housing.csv')

# Step2.2: Print first few rows of this data

print('Print first few rows of this data - ')
print()
print(housingData.head())

# Step2.3: Extract input (X) and output (y) data from the dataset

X = housingData.iloc[:, :-1].values
y = housingData.iloc[:, [-1]].values
```

```
# Step3: Handle missing values:
# Fill the missing values with the mean of the respective column

from sklearn.preprocessing import Imputer
missingValueImputer = Imputer()
X[:, :-1] = missingValueImputer.fit_transform(X[:, :-1])
y = missingValueImputer.fit_transform(y)

# Step4: Encode categorical data:
# Convert categorical column in the dataset to numerical data

from sklearn.preprocessing import LabelEncoder
X_labelencoder = LabelEncoder()
X[:, -1] = X_labelencoder.fit_transform(X[:, -1])

# Step5: Split the dataset: Split the data into
# 80% training dataset and 20% test dataset

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 = 0)

# Step6: Standardize data: Standardize training and test datasets

from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
y_train = scaler.fit_transform(y_train)
y_test = scaler.transform(y_test)

####################################################################
'''Task1: Perform Linear Regression'''
####################################################################

# Task1.1: Perform Linear Regression on training data

from sklearn.linear_model import LinearRegression
linearRegression = LinearRegression()
linearRegression.fit(X_train, y_train)

# Task1.2: Predict output for test dataset using the fitted model

predictionLinear = linearRegression.predict(X_test)

# Task1.3: Print root mean squared error (RMSE) from Linear Regression

from sklearn.metrics import mean_squared_error
mseLinear = mean_squared_error(y_test, predictionLinear)
print('Root mean squared error (RMSE) from Linear Regression = ')
print(mseLinear)

#################################################################
####
'''Task2: Perform Decision Tree Regression'''
#################################################################
####

# Task2.1: Perform Decision Tree Regression on training data

from sklearn.tree import DecisionTreeRegressor
DTregressor = DecisionTreeRegressor()
DTregressor.fit(X_train, y_train)

# Task2.2: Predict output for test dataset using the fitted model

predictionDT = DTregressor.predict(X_test)

# Task2.3: Print root mean squared error from Decision Tree Regression

from sklearn.metrics import mean_squared_error
mseDT = mean_squared_error(y_test, predictionDT)
print('Root mean squared error from Decision Tree Regression = ')
print(mseDT)

#################################################################
####
```

```
'''Task3: Perform Random Forest Regression'''
#################################################################
####

# Task3.1: Perform Random Forest Regression on training data

from sklearn.ensemble import RandomForestRegressor
RFregressor = RandomForestRegressor()
RFregressor.fit(X_train, y_train)

# Task3.2: Predict output for test dataset using the fitted model

predictionRF = RFregressor.predict(X_test)

# Task3.3: Print root mean squared error from Random Forest Regression

from sklearn.metrics import mean_squared_error
mseRF = mean_squared_error(y_test, predictionRF)
print('Root mean squared error from Random Forest Regression = ')
print(mseRF)

#################################################################
####
'''Task4: Bonus exercise:
   Perform Linear Regression with one independent variable'''
#################################################################
####
```

```
# Task4.1: Extract just the median_income column from the
# independent variables (from X_train and X_test)

X_train_median_income = X_train[: , [7]]
X_test_median_income = X_test[: , [7]]

# Task4.2: Perform Linear Regression to predict housing values
# based on median_income

from sklearn.linear_model import LinearRegression
linearRegression2 = LinearRegression()
linearRegression2.fit(X_train_median_income, y_train)

# Task4.3: Predict output for test dataset using the fitted model

predictionLinear2 = linearRegression2.predict(X_test_median_income)

# Task4.4: Plot the fitted model for training data as well as
# for test data to check if the fitted model satisfies the test data

# Task4.4.1: let us visualize the Training set

plt.scatter(X_train_median_income, y_train, color = 'green')
plt.plot (X_train_median_income,
        linearRegression2.predict(X_train_median_income), color = 'red')
```

```
plt.title ('compare Training result - median_income / median_house_value')
plt.xlabel('median_income')
plt.ylabel('median_house_value')
plt.show()

# Task4.4.2: let us visualize the Testing set

plt.scatter(X_test_median_income, y_test, color = 'blue')
plt.plot (X_train_median_income,
        linearRegression2.predict(X_train_median_income), color = 'red')
plt.title ('compare Testing result - median_income / median_house_value')
plt.xlabel('median_income')
plt.ylabel('median_house_value')
plt.show()

#######################################################################
####
'''                    End                    '''
#######################################################################
####
```

Code Snippet followed a Screenshot of the output

Q1. Load the data and Print first few rows of this data

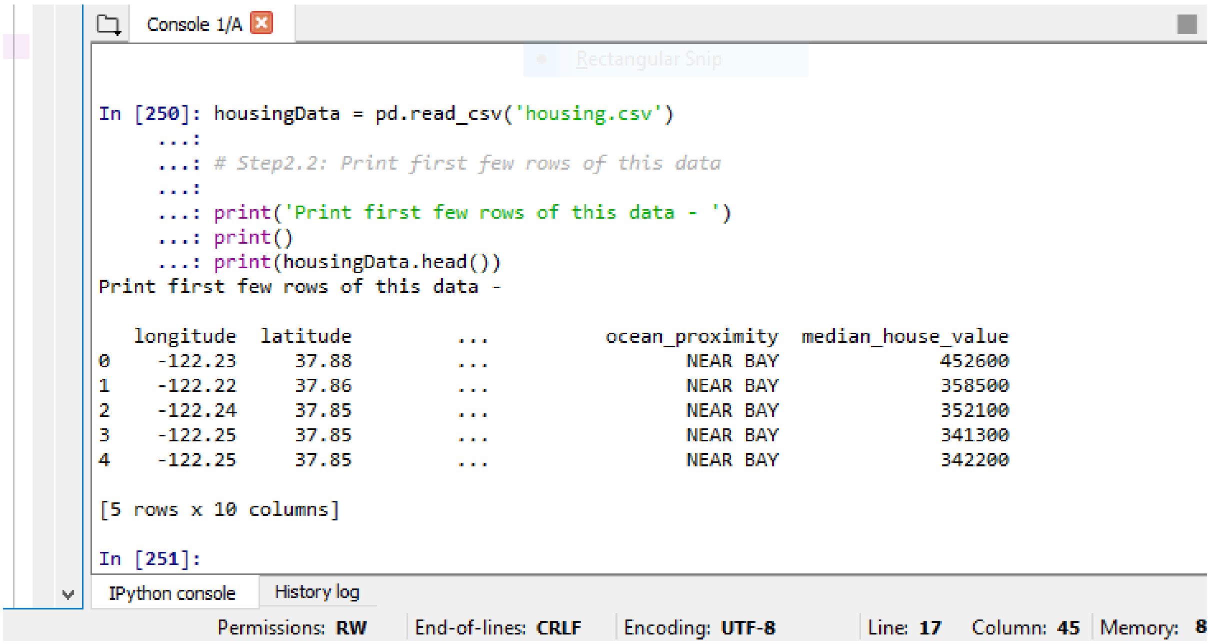
```
# Step2: Load the data
# Step2.1: Read the “housing.csv” file from the folder into the program
housingData = pd.read_csv('housing.csv')
```

Step2.2: Print first few rows of this data

print('Print first few rows of this data - ')

print()

print(housingData.head())



```
In [250]: housingData = pd.read_csv('housing.csv')
     ...:
     ...: # Step2.2: Print first few rows of this data
     ...:
     ...: print('Print first few rows of this data - ')
     ...: print()
     ...: print(housingData.head())
Print first few rows of this data -

   longitude  latitude         ...          ocean_proximity  median_house_value
0    -122.23     37.88         ...                 NEAR BAY              452600
1    -122.22     37.86         ...                 NEAR BAY              358500
2    -122.24     37.85         ...                 NEAR BAY              352100
3    -122.25     37.85         ...                 NEAR BAY              341300
4    -122.25     37.85         ...                 NEAR BAY              342200

[5 rows x 10 columns]

In [251]:
```

Permissions: RW   End-of-lines: CRLF   Encoding: UTF-8   Line: 17   Column: 45   Memory: 8

Q2. Separate features and labels, deal with missing value, Encode categorical data, Split the dataset into training and testing set, standardized the data

# Step2.3: Extract input (X) and output (y) data from the dataset

X = housingData.iloc[:, :-1].values

y = housingData.iloc[:, [-1]].values

# Step3: Handle missing values:

# Fill the missing values with the mean of the respective column

from sklearn.preprocessing import Imputer

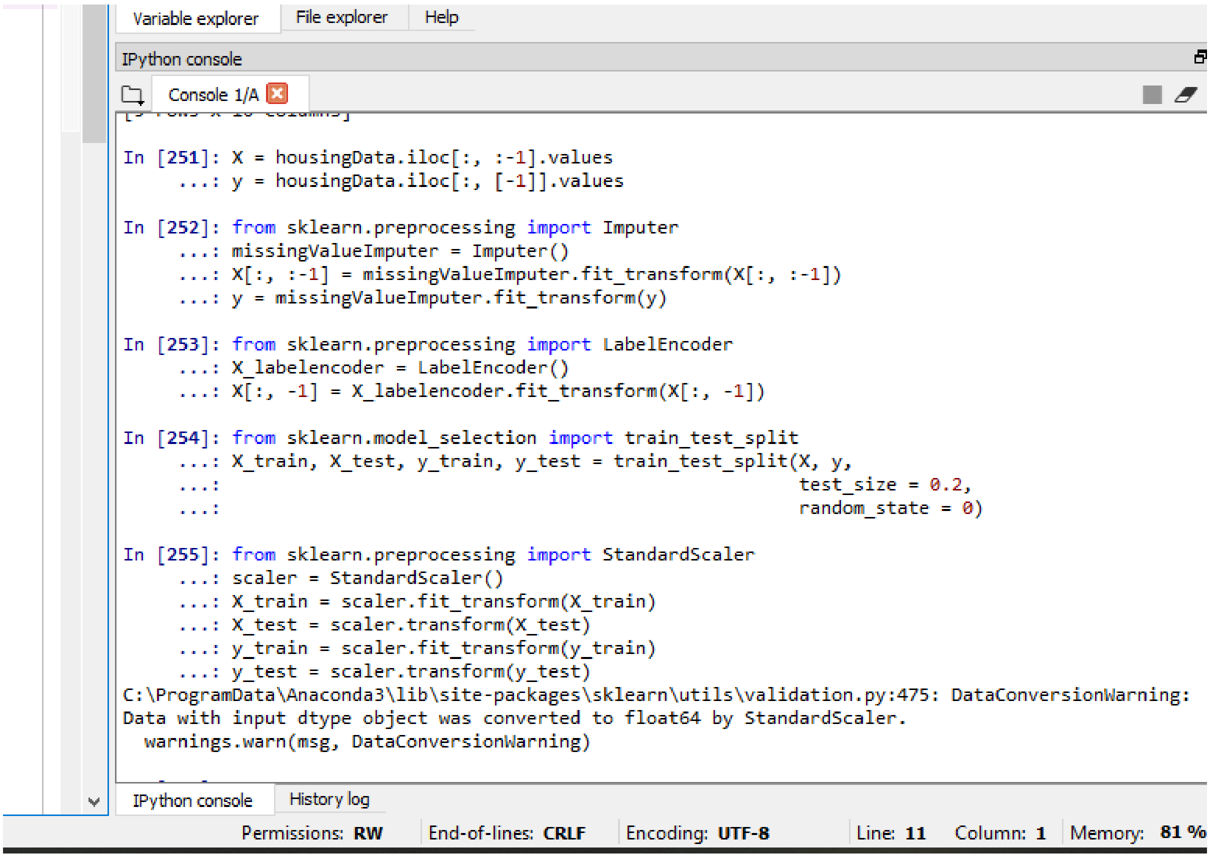
missingValueImputer = Imputer()

X[:, :-1] = missingValueImputer.fit_transform(X[:, :-1])

y = missingValueImputer.fit_transform(y)

# Step4: Encode categorical data:

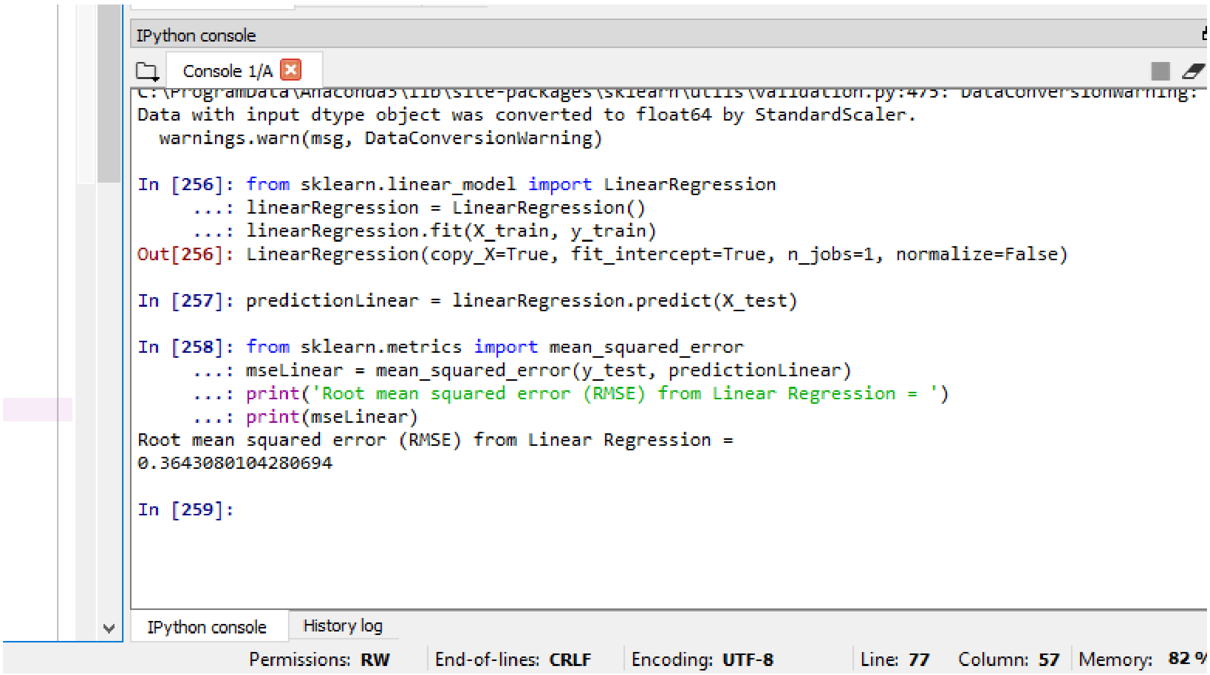
```
# Convert categorical column in the dataset to numerical data
from sklearn.preprocessing import LabelEncoder
X_labelencoder = LabelEncoder()
X[:, -1] = X_labelencoder.fit_transform(X[:, -1])
# Step5: Split the dataset: Split the data into
# 80% training dataset and 20% test dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                 test_size = 0.2,
                                 random_state = 0)
# Step6: Standardize data: Standardize training and test datasets
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
y_train = scaler.fit_transform(y_train)
y_test = scaler.transform(y_test)
```



```
In [251]: X = housingData.iloc[:, :-1].values
     ...: y = housingData.iloc[:, [-1]].values

In [252]: from sklearn.preprocessing import Imputer
     ...: missingValueImputer = Imputer()
     ...: X[:, :-1] = missingValueImputer.fit_transform(X[:, :-1])
     ...: y = missingValueImputer.fit_transform(y)

In [253]: from sklearn.preprocessing import LabelEncoder
     ...: X_labelencoder = LabelEncoder()
     ...: X[:, -1] = X_labelencoder.fit_transform(X[:, -1])

In [254]: 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 = 0)

In [255]: from sklearn.preprocessing import StandardScaler
     ...: scaler = StandardScaler()
     ...: X_train = scaler.fit_transform(X_train)
     ...: X_test = scaler.transform(X_test)
     ...: y_train = scaler.fit_transform(y_train)
     ...: y_test = scaler.transform(y_test)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype object was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

Q3. Task1: Perform Linear Regression

```
# Task1.1: Perform Linear Regression on training data

from sklearn.linear_model import LinearRegression

linearRegression = LinearRegression()

linearRegression.fit(X_train, y_train)

# Task1.2: Predict output for test dataset using the fitted model

predictionLinear = linearRegression.predict(X_test)

# Task1.3: Print root mean squared error (RMSE) from Linear Regression

from sklearn.metrics import mean_squared_error

mseLinear = mean_squared_error(y_test, predictionLinear)

print('Root mean squared error (RMSE) from Linear Regression = ')

print(mseLinear)
```

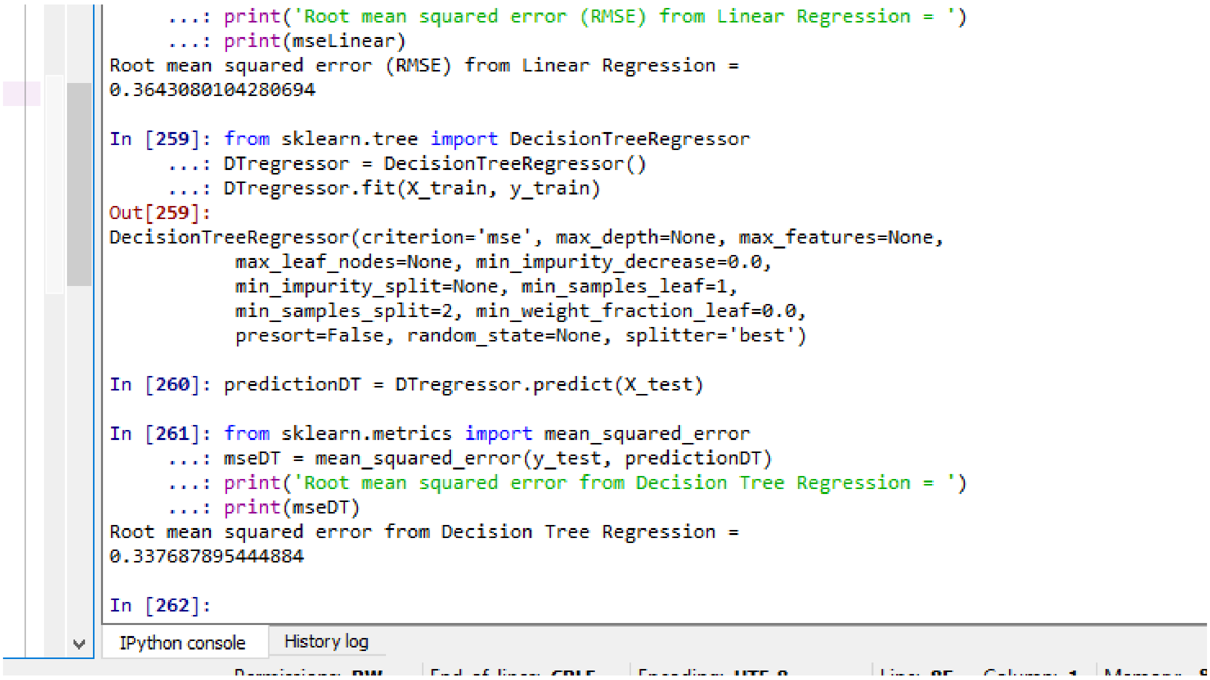


```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype object was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)

In [256]: from sklearn.linear_model import LinearRegression
     ...: linearRegression = LinearRegression()
     ...: linearRegression.fit(X_train, y_train)
Out[256]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [257]: predictionLinear = linearRegression.predict(X_test)

In [258]: from sklearn.metrics import mean_squared_error
     ...: mseLinear = mean_squared_error(y_test, predictionLinear)
     ...: print('Root mean squared error (RMSE) from Linear Regression = ')
     ...: print(mseLinear)
Root mean squared error (RMSE) from Linear Regression =
0.3643080104280694

In [259]:
```

Permissions: RW   End-of-lines: CRLF   Encoding: UTF-8   Line: 77   Column: 57   Memory: 82 %

Q4. Task2: Perform Decision Tree Regression

```
# Task2.1: Perform Decision Tree Regression on training data

from sklearn.tree import DecisionTreeRegressor

DTregressor = DecisionTreeRegressor()

DTregressor.fit(X_train, y_train)

# Task2.2: Predict output for test dataset using the fitted model

predictionDT = DTregressor.predict(X_test)

# Task2.3: Print root mean squared error from Decision Tree Regression

from sklearn.metrics import mean_squared_error

mseDT = mean_squared_error(y_test, predictionDT)

print('Root mean squared error from Decision Tree Regression = ')

print(mseDT)
```



```
     ...: print('Root mean squared error (RMSE) from Linear Regression = ')
     ...: print(mseLinear)
Root mean squared error (RMSE) from Linear Regression =
0.3643080104280694

In [259]: from sklearn.tree import DecisionTreeRegressor
     ...: DTregressor = DecisionTreeRegressor()
     ...: DTregressor.fit(X_train, y_train)
Out[259]:
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')

In [260]: predictionDT = DTregressor.predict(X_test)

In [261]: from sklearn.metrics import mean_squared_error
     ...: mseDT = mean_squared_error(y_test, predictionDT)
     ...: print('Root mean squared error from Decision Tree Regression = ')
     ...: print(mseDT)
Root mean squared error from Decision Tree Regression =
0.337687895444884

In [262]:
```

Q5. Task3: Perform Random Forest Regression

# Task3.1: Perform Random Forest Regression on training data

from sklearn.ensemble import RandomForestRegressor

RFregressor = RandomForestRegressor()

RFregressor.fit(X_train, y_train)

# Task3.2: Predict output for test dataset using the fitted model

predictionRF = RFregressor.predict(X_test)

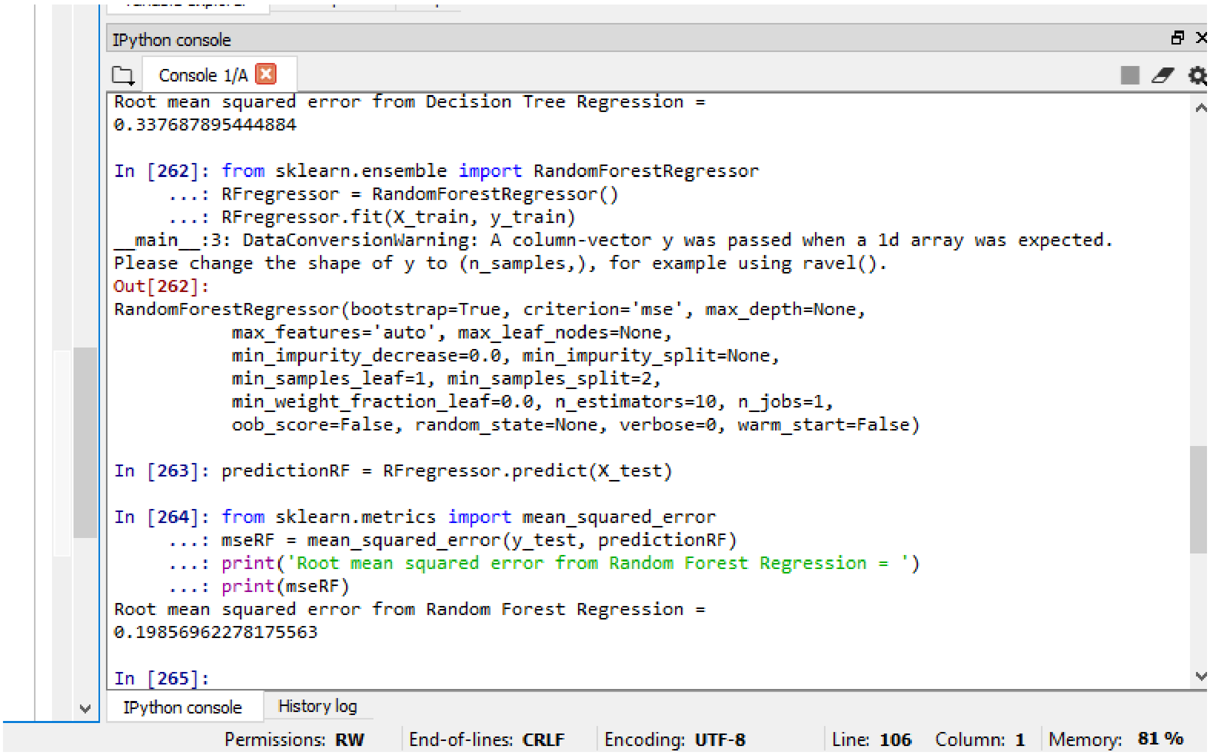
# Task3.3: Print root mean squared error from Random Forest Regression

from sklearn.metrics import mean_squared_error

mseRF = mean_squared_error(y_test, predictionRF)

print('Root mean squared error from Random Forest Regression = ')

print(mseRF)



```
Root mean squared error from Decision Tree Regression =
0.337687895444884

In [262]: from sklearn.ensemble import RandomForestRegressor
     ...: RFregressor = RandomForestRegressor()
     ...: RFregressor.fit(X_train, y_train)
__main__:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples,), for example using ravel().
Out[262]:
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)

In [263]: predictionRF = RFregressor.predict(X_test)

In [264]: from sklearn.metrics import mean_squared_error
     ...: mseRF = mean_squared_error(y_test, predictionRF)
     ...: print('Root mean squared error from Random Forest Regression = ')
     ...: print(mseRF)
Root mean squared error from Random Forest Regression =
0.19856962278175563

In [265]:
```

Q6. Task4: Bonus exercise: Perform Linear Regression with one independent variable

# Task4.1: Extract just the median_income column from the

# independent variables (from X_train and X_test)

X_train_median_income = X_train[: , [7]]

X_test_median_income = X_test[: , [7]]

# Task4.2: Perform Linear Regression to predict housing values

# based on median_income

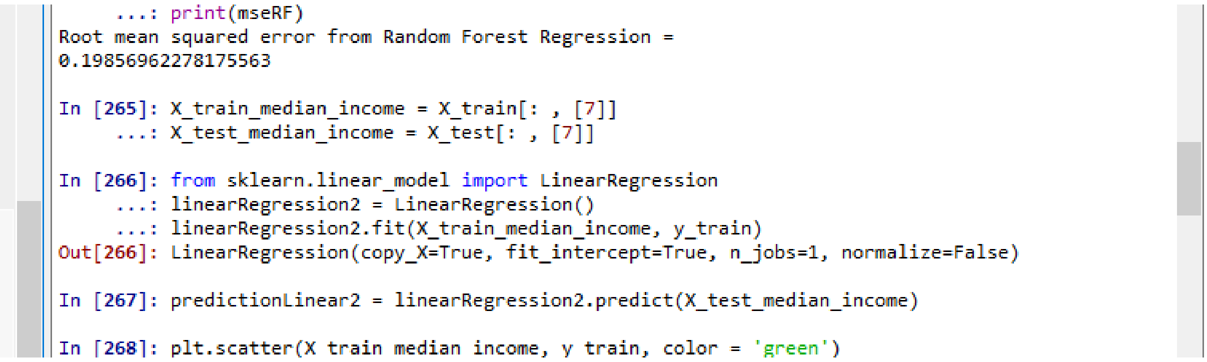
from sklearn.linear_model import LinearRegression

linearRegression2 = LinearRegression()

linearRegression2.fit(X_train_median_income, y_train)

# Task4.3: Predict output for test dataset using the fitted model

predictionLinear2 = linearRegression2.predict(X_test_median_income)



```
     ...: print(mseRF)
Root mean squared error from Random Forest Regression =
0.19856962278175563

In [265]: X_train_median_income = X_train[: , [7]]
     ...: X_test_median_income = X_test[: , [7]]

In [266]: from sklearn.linear_model import LinearRegression
     ...: linearRegression2 = LinearRegression()
     ...: linearRegression2.fit(X_train_median_income, y_train)
Out[266]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [267]: predictionLinear2 = linearRegression2.predict(X_test_median_income)

In [268]: plt.scatter(X train median income, y train, color = 'green')
```

# Task4.4: Plot the fitted model for training data as well as

# for test data to check if the fitted model satisfies the test data

# Task4.4.1: **let us visualize the Training set**

plt.scatter(X_train_median_income, y_train, color = 'green')

plt.plot (X_train_median_income,

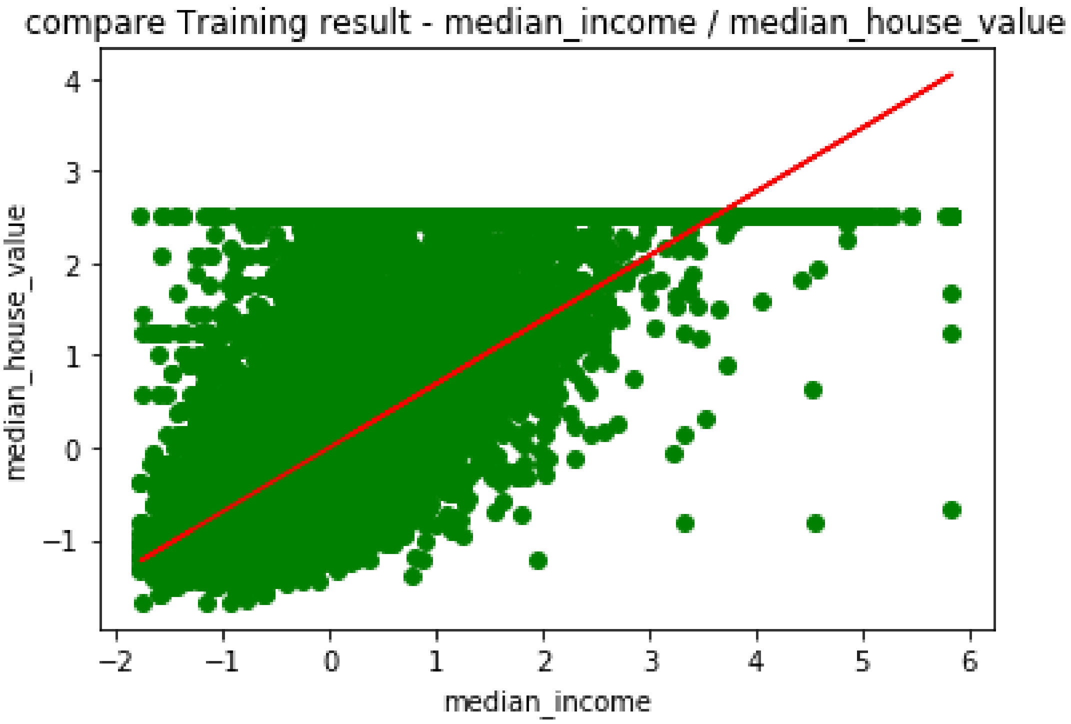
        linearRegression2.predict(X_train_median_income), color = 'red')

plt.title ('compare Training result - median_income / median_house_value')

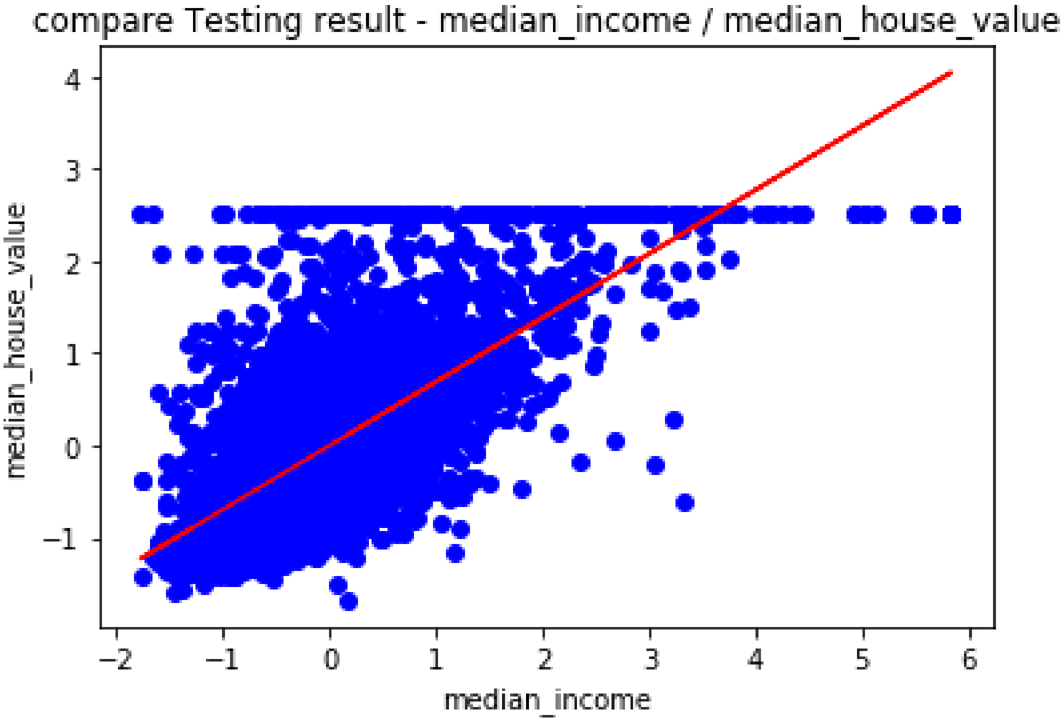
plt.xlabel('median_income')

plt.ylabel('median_house_value')

plt.show()



```
# Task4.4.2: let us visualize the Testing set
plt.scatter(X_test_median_income, y_test, color = 'blue')
plt.plot (X_train_median_income,
        linearRegression2.predict(X_train_median_income), color = 'red')
plt.title ('compare Testing result - median_income / median_house_value')
plt.xlabel('median_income')
plt.ylabel('median_house_value')
plt.show()
```


compare Testing result - median_income / median_house_value
median_house_value
median_income
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6

****End ****