Deniz Aytemiz

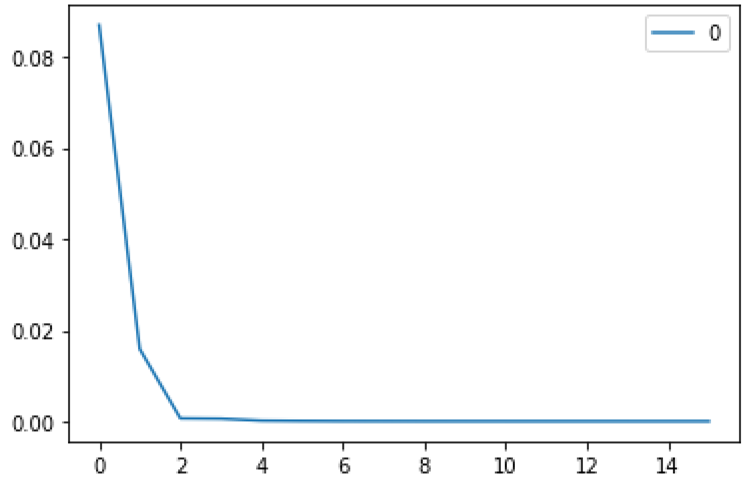
ECE 5984-HW6

The features used are, the passenger count, pick up county, time passed/ duration of the trip, passenger count, trip distance, improvement surcharge and pick up county. Pick up county is not dropped since the charge per mile may be different for each county. The time passed in terms of hours are found by the pick up and drop of times subtraction. The rest of the features are dropped. Total amount is the target variable. All features and the target are normalized in the rest of the work.
The pick up county feature is categorical therefore in order to include it, it is one hot encoded.

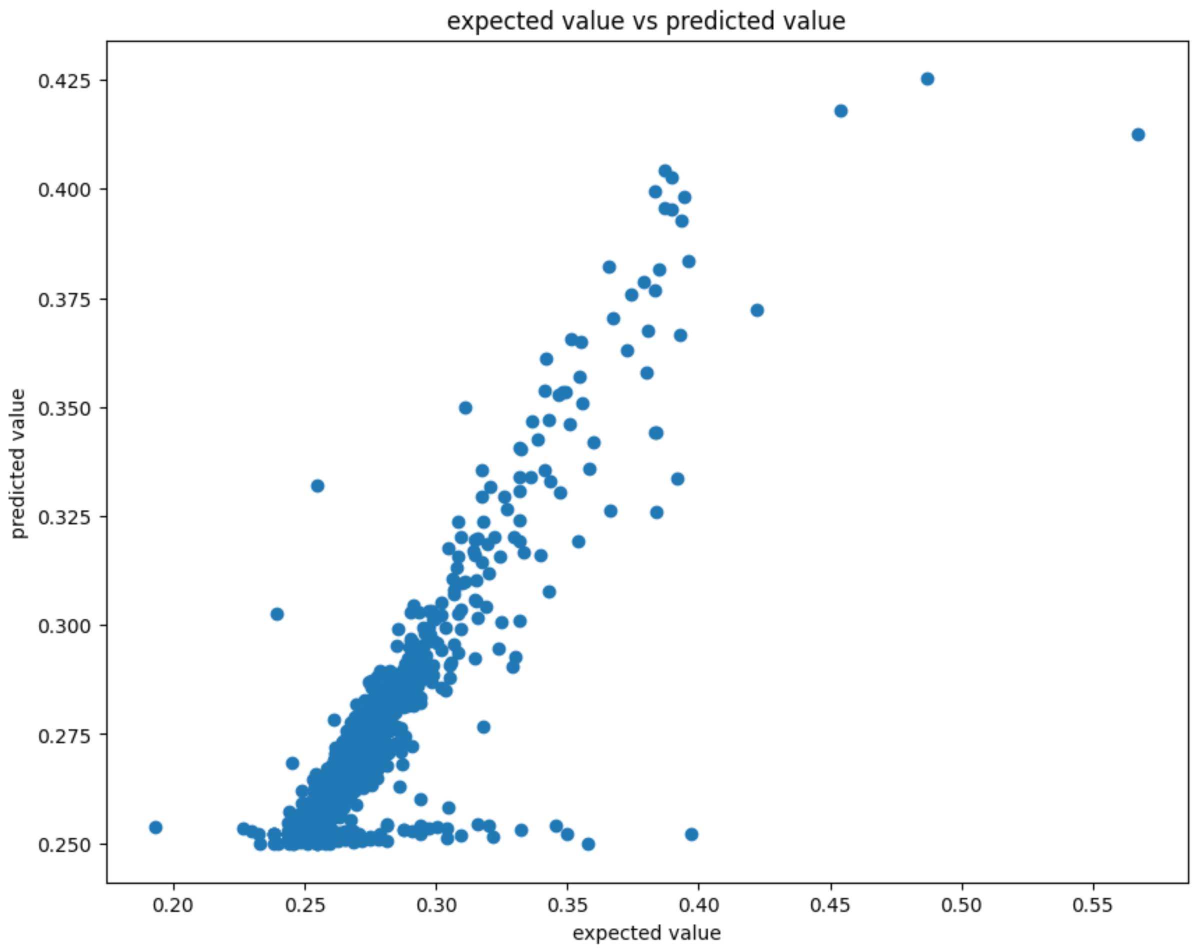
The three first stage models are; MLP Regressor, Decision Tree Regressor and Multivariate Linear Regressor. The second stage model is MLP regressor.
Each first stage models' MSE and R2 are displayed in the Table below.

	MLP classifier	Decision Tree	Linear Regression
MSE	0.0003926	0.00038332	0.000374615
R2	0.0625476	0.07907461	0.098597831

In the 1st stage time difference and trip distance features are not included in the dataset. The expected output is calculated at the first stage without these variables. Then at the next stage this output, trip distance and duration of the trip are the three features used in the second model.
The learning curve of the 1st stage and 2nd stage MLP model is displayed in Figure below.



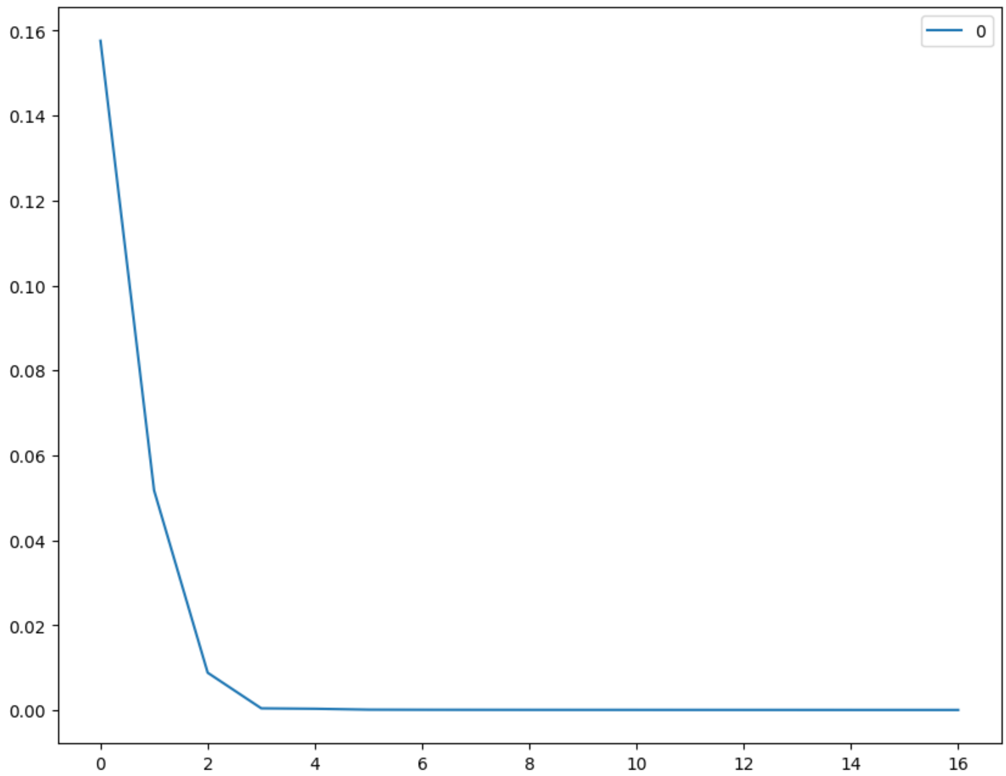
The expected and actual values are plotted in the scatter plot below. It is observed that there is a linear relation.



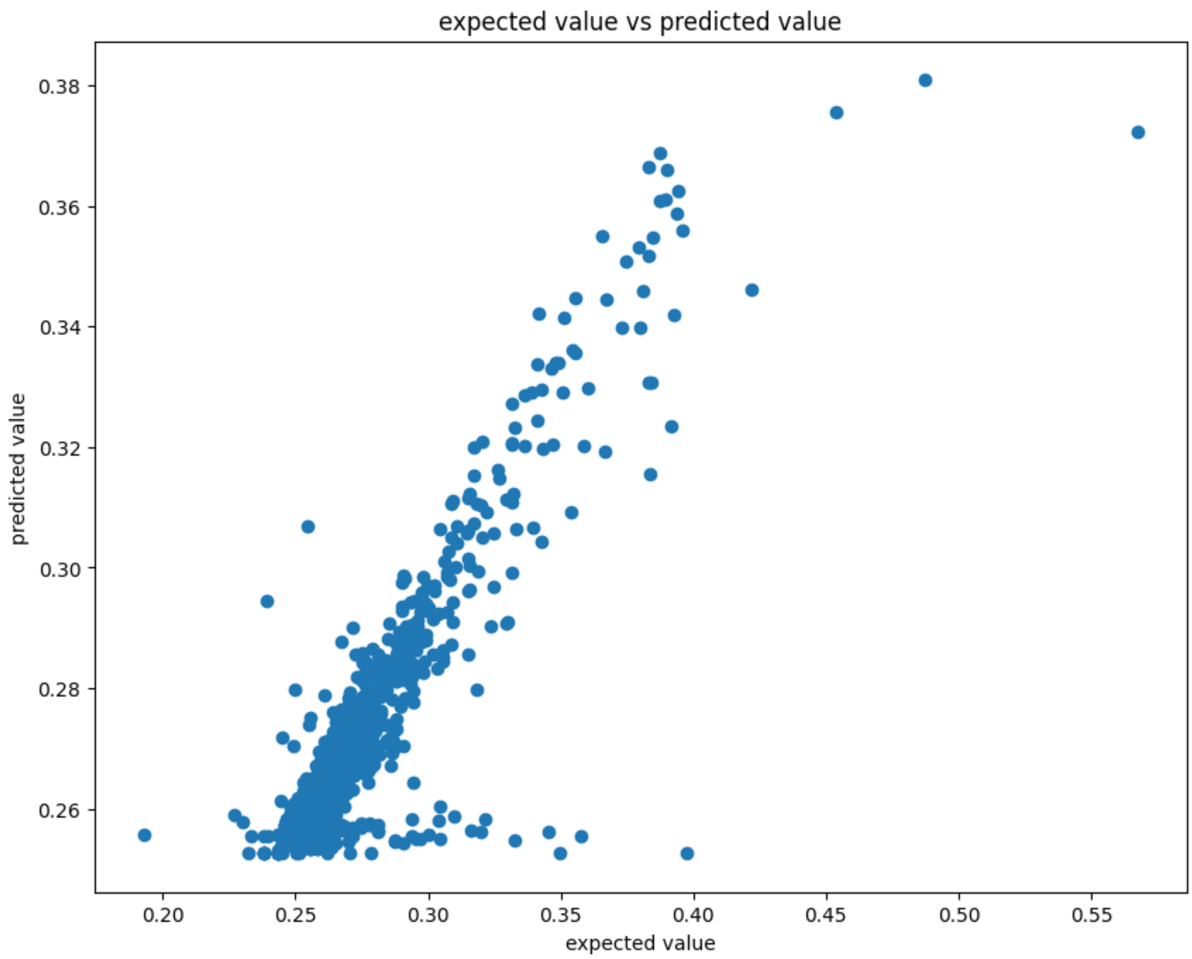
The performance metrics are given in the table below.

	1st stage: MLP classifier & 2nd stage: MLP classifier
MSE	9.04E-05
R2	0.799085815
MAE	0.006482686
EVS	0.799095598

The learning curve of the 1st stage Decision tree and 2nd stage MLP model is displayed in Figure below.



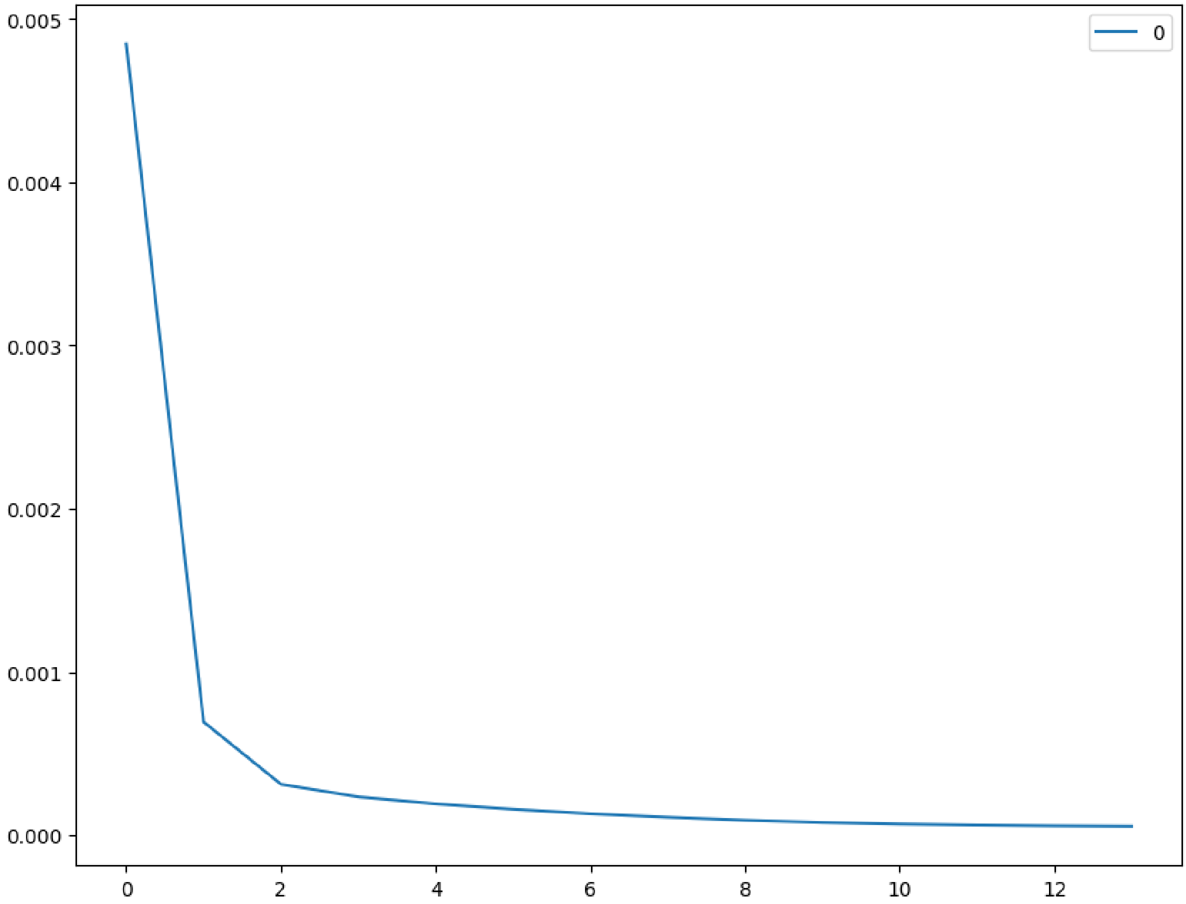
The expected and actual values are plotted in the scatter plot below.



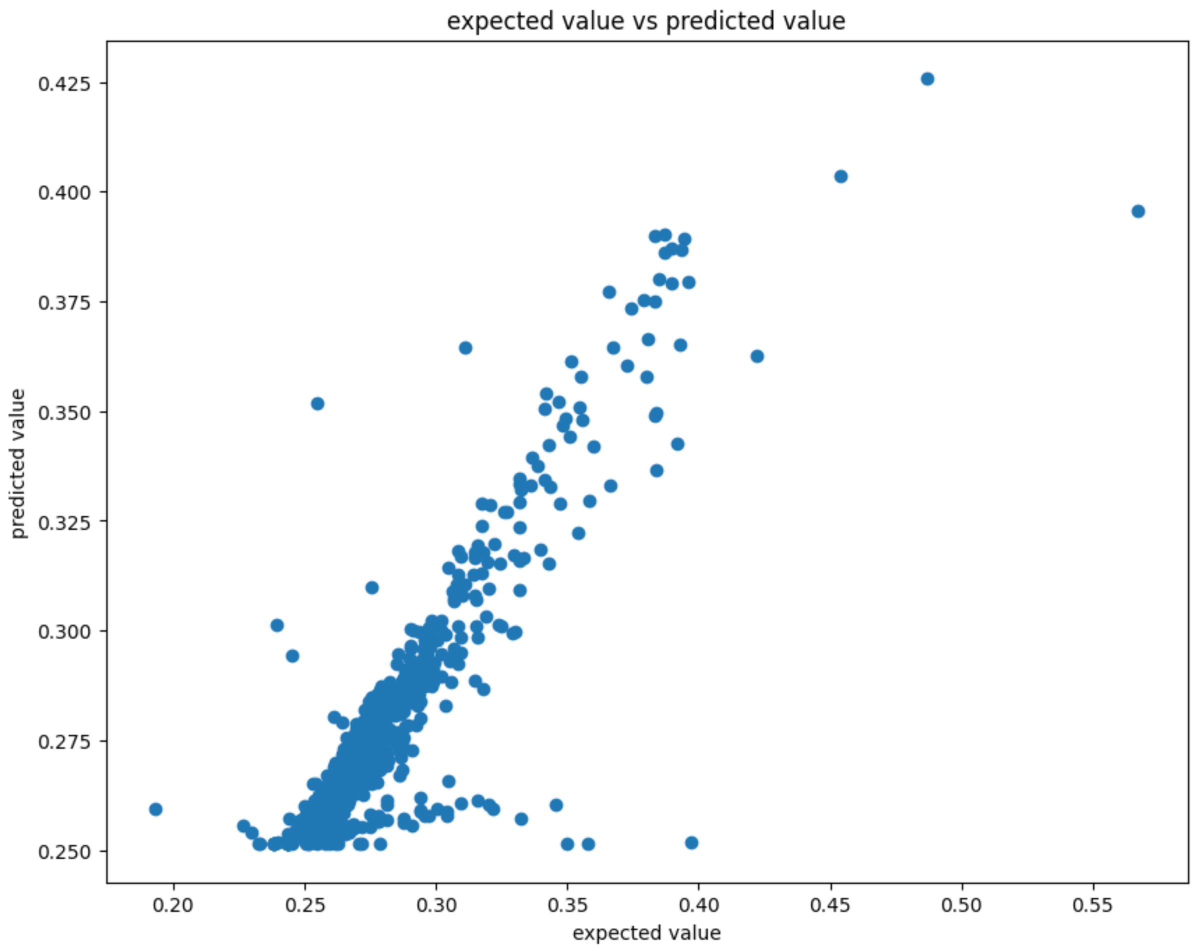
The performance metrics are given in the table below.

	1st stage:Decision Tree & 2nd stage: MLP classifier
MSE	0.000114504
R2	0.741293576
MAE	0.007946733
EVS	0.741298941

The learning curve of the 1st stage Linear Regression and 2nd stage MLP model is displayed in Figure below.



The expected and actual values are plotted in the scatter plot below.



The performance metrics are given in the table below.

	1st stage:Linear regression & 2nd stage: MLP classifier
MSE	8.67E-05
R2	0.805467429
MAE	0.006242225
EVS	0.805483967

All models performing metrics together are given below:

	1st stage: MLP classifier & 2nd stage: MLP classifier	1st stage:Decision Tree & 2nd stage: MLP classifier	1st stage:Linear regression & 2nd stage: MLP classifier
MSE	9.04E-05	0.000114504	8.67E-05
R2	0.799085815	0.741293576	0.805467429
MAE	0.006482686	0.007946733	0.006242225
EVS	0.799095598	0.741298941	0.805483967

It seems like the worst performing model is Decision Tree and MLP. I would say the best performing model is the 3rd one with Linear Regression and MLP regressor since it has the lowest MSE and MAE and highest R2 and EVS values. I think this is mostly because the linear regression model is performing well. If the 2nd stage model would be chosen as multivariate linear regressor each model might have performed better.

Code:
```
#load data
import pandas as pd
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import numpy as np


df = pd.read_excel('Taxi_Trip_Data.xlsx')

#drop variables that are not going to be feautures
df = df.drop(['store_and_fwd_flag', 'PULocationID','DOLocationID'],
axis=1)
df = df.drop(['fare_amount',
'extra','mta_tax','tip_amount','tolls_amount','DOBorough'], axis=1)
df
#take time difference in hours
df['Difference'] = (df['lpep_dropoff_datetime'] -
df['lpep_pickup_datetime'])
df['Difference'] = df['Difference'].dt.seconds
df['Difference'] =df['Difference']/60
#normalize data
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
```

```
df[['passenger_count',
'trip_distance','improvement_surcharge','Difference','total_amount']] =
min_max_scaler.fit_transform(df[['passenger_count',
'trip_distance','improvement_surcharge','Difference','total_amount']])
df
#one hot encode the pick up county since it is categorical
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(handle_unknown='ignore')
print(df['PUBorough'].unique())
new_df =
pd.DataFrame(encoder.fit_transform(df[['PUBorough']]).toarray())
new_df.columns = ['bronx', 'brooklyn', 'manhattan', 'queens', 'staten
island','unknown']
final_df=pd.concat([df, new_df],axis=1)
final_df
# trip_distance and difference will be feautures in second stage model
so we drop them too
from sklearn.model_selection import train_test_split
X = final_df
y = final_df["total_amount"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=1)
# dropping unused variables from train and test sets respectively
X_train_1= X_train[['trip_distance' , 'Difference','total_amount']]
X_train= X_train.drop('lpep_pickup_datetime',axis=1)
X_train= X_train.drop('lpep_dropoff_datetime',axis=1)
X_train= X_train.drop('RatecodeID',axis=1)
X_train= X_train.drop('PUBorough',axis=1)
X_train= X_train.drop('trip_distance',axis=1)
X_train= X_train.drop('Difference',axis=1)
X_train= X_train.drop('VendorID',axis=1)
X_train= X_train.drop('total_amount',axis=1)

X_train

X_test_1=X_test[['trip_distance' , 'Difference','total_amount']]
X_test= X_test.drop('lpep_pickup_datetime',axis=1)
X_test= X_test.drop('lpep_dropoff_datetime',axis=1)
X_test= X_test.drop('RatecodeID',axis=1)
X_test= X_test.drop('PUBorough',axis=1)
X_test= X_test.drop('trip_distance',axis=1)
X_test= X_test.drop('Difference',axis=1)
X_test= X_test.drop('VendorID',axis=1)
```

```
X_test= X_test.drop('total_amount',axis=1)

X_test
#this df is saved for 2nd stage modeling. The output of the 1st input
will be added
#to this df as another column
X_test_1
#1st stage model: regression neural network
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
from sklearn import metrics
model = MLPRegressor()
model.fit(X_train, y_train)
print(model)

expected_y  = y_test
predicted_y = model.predict(X_test)
print(metrics.r2_score(expected_y, predicted_y))
print(metrics.mean_squared_log_error(expected_y, predicted_y))
#1st stage model: regression decision tree

from sklearn.tree import DecisionTreeRegressor
tree = DecisionTreeRegressor(max_depth=2)
tree.fit(X_train, y_train)
y_pred = tree.predict(X_test)
expected_y  = y_test
print(metrics.r2_score(expected_y, y_pred))
print(metrics.mean_squared_log_error(expected_y, y_pred))
#1st stage model: regression multivariant linear
from sklearn.linear_model import LinearRegression
# creating an object of LinearRegression class
LR = LinearRegression()
# fitting the training data
LR.fit(X_train,y_train)
y_prediction =  LR.predict(X_test)
print(y_prediction)
expected_y  = y_test
print(metrics.r2_score(expected_y, y_prediction))
print(metrics.mean_squared_log_error(expected_y, y_prediction))
#2nd stage model:MLP regressor:
#1st stage model:MLP regressor:

MLP_df = pd.DataFrame(predicted_y, columns = ['1st_output_MLP'])
```

```
MLP_df
#final_df1=pd.concat([MLP_df, X_test],axis=1)
X_test_MLP = X_test_1
X_test_MLP['1st_output'] = MLP_df['1st_output_MLP']
X_test_MLP = X_test_MLP[X_test_MLP['1st_output'].notna()]
X_test_MLP
#2nd stage model MLP Regressor
X = X_test_MLP
X = X.drop('total_amount',axis=1)

y = X_test_MLP["total_amount"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=2)
model.fit(X_train, y_train)
print(model)
expected_y  = y_test
predicted_y = model.predict(X_test)
print(metrics.r2_score(expected_y, predicted_y))
print(metrics.mean_squared_log_error(expected_y, predicted_y))
print(metrics.explained_variance_score(expected_y, predicted_y))
print(mean_absolute_error(expected_y, predicted_y))
pd.DataFrame(model.loss_curve_).plot()
plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})
plt.scatter(expected_y, predicted_y)
plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})
plt.title('expected value vs predicted value')
plt.xlabel('expected value')
plt.ylabel('predicted value')
plt.show()
#2nd stage model:MLP regressor:
#1st stage model:Decision tree regressor:

#X_test_tree is the dataset we are going to use for this modelling
tree_df = pd.DataFrame(y_pred, columns = ['1st_output_tree'])
X_test_tree = X_test_1
X_test_tree['1st_output'] = tree_df['1st_output_tree']
X_test_tree = X_test_tree[X_test_tree['1st_output'].notna()]
X_test_tree

X = X_test_tree
X = X.drop('total_amount',axis=1)

y = X_test_tree["total_amount"]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=2)
model.fit(X_train, y_train)
print(model)
expected_y  = y_test
predicted_y = model.predict(X_test)
print(metrics.r2_score(expected_y, predicted_y))
print(metrics.mean_squared_log_error(expected_y, predicted_y))
print(metrics.explained_variance_score(expected_y, predicted_y))
print(mean_absolute_error(expected_y, predicted_y))
pd.DataFrame(model.loss_curve_).plot()

plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})
plt.scatter(expected_y, predicted_y)
plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})
plt.title('expected value vs predicted value')
plt.xlabel('expected value')
plt.ylabel('predicted value')
plt.show()
#3rd stage model:Multivariate Linear regressor:
#1st stage model:Decision tree regressor:

#X_test_tree is the dataset we are going to use for this modelling
linReg_df = pd.DataFrame(y_prediction, columns = ['1st_output_tree'])
X_test_linReg = X_test_1
X_test_linReg['1st_output'] = linReg_df['1st_output_tree']
X_test_linReg = X_test_linReg[X_test_linReg['1st_output'].notna()]
X_test_linReg

X = X_test_linReg
X = X.drop('total_amount',axis=1)

y = X_test_linReg["total_amount"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=2)
model.fit(X_train, y_train)
print(model)
expected_y  = y_test
predicted_y = model.predict(X_test)
print(metrics.r2_score(expected_y, predicted_y))
print(metrics.mean_squared_log_error(expected_y, predicted_y))
print(metrics.explained_variance_score(expected_y, predicted_y))
print(mean_absolute_error(expected_y, predicted_y))
```

```
pd.DataFrame(model.loss_curve_).plot()
plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})
plt.scatter(expected_y, predicted_y)
plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})
plt.title('expected value vs predicted value')
plt.xlabel('expected value')
plt.ylabel('predicted value')
plt.show()
```