Solution_Assignment4

September 30, 2022

1 Step 1: import all the necessary libraries.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
# activate plot theme
import qeds

qeds.themes.mpl_style();
plotly_template = qeds.themes.plotly_template()
colors = qeds.themes.COLOR_CYCLE

from sklearn import (linear_model, metrics, model_selection)
```

[2]:
```
df= pd.read_csv('Airfares.csv')
```

Explore the numerical predictors and outcome (FARE) by creating a correlation table, heat map, and examining some scatterplots between FARE and those predictors. What seems to be the best single predictor of FARE?

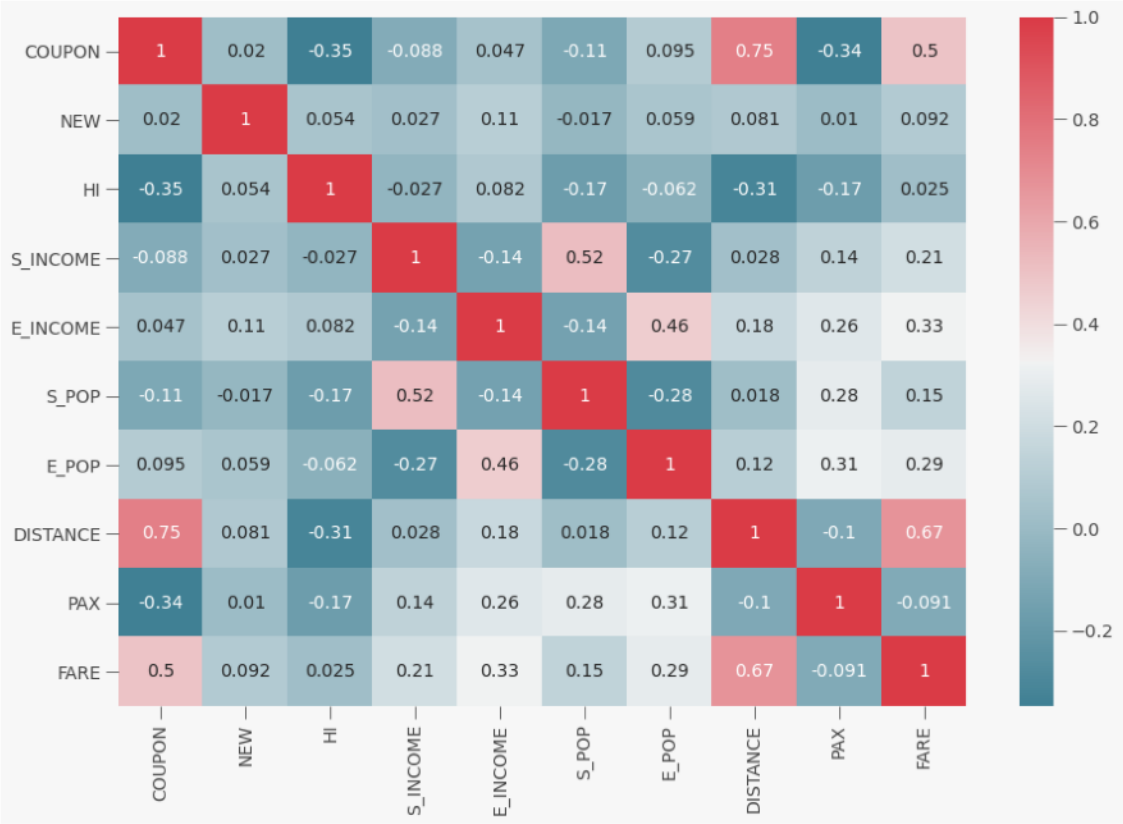
[3]:
```
correlation_data = df.corr()
correlation_data
```

[3]:

	COUPON	NEW	HI	S_INCOME	E_INCOME	S_POP	\
COUPON	1.000000	0.020223	-0.347252	-0.088403	0.046889	-0.107763	
NEW	0.020223	1.000000	0.054147	0.026597	0.113377	-0.016672	
HI	-0.347252	0.054147	1.000000	-0.027382	0.082393	-0.172495	
S_INCOME	-0.088403	0.026597	-0.027382	1.000000	-0.138864	0.517187	
E_INCOME	0.046889	0.113377	0.082393	-0.138864	1.000000	-0.144059	
S_POP	-0.107763	-0.016672	-0.172495	0.517187	-0.144059	1.000000	
E_POP	0.094970	0.058568	-0.062456	-0.272280	0.458418	-0.280143	
DISTANCE	0.746805	0.080965	-0.312375	0.028153	0.176531	0.018437	
PAX	-0.336974	0.010495	-0.168961	0.138197	0.259961	0.284611	
FARE	0.496537	0.091730	0.025195	0.209135	0.326092	0.145097	

	E_POP	DISTANCE	PAX	FARE
COUPON	0.094970	0.746805	-0.336974	0.496537
NEW	0.058568	0.080965	0.010495	0.091730
HI	-0.062456	-0.312375	-0.168961	0.025195
S_INCOME	-0.272280	0.028153	0.138197	0.209135
E_INCOME	0.458418	0.176531	0.259961	0.326092
S_POP	-0.280143	0.018437	0.284611	0.145097
E_POP	1.000000	0.115640	0.314698	0.285043
DISTANCE	0.115640	1.000000	-0.102482	0.670016
PAX	0.314698	-0.102482	1.000000	-0.090705
FARE	0.285043	0.670016	-0.090705	1.000000

[4]:
```
_,ax=plt.subplots(figsize=(15,10))
colormap=sns.diverging_palette(220,10,as_cmap=True)
sns.heatmap(df.corr(),annot=True,cmap=colormap)
plt.savefig("correlation.jpeg")
```



As shown in the above heat map, the air fare is highly positively with the distance and coupon. We then can plot its relationship by using the scatter plots as in the below graphs.

[5]:
```
# notice the log here!
y = np.log(df["FARE"]) #ln(price)
df["log_price"] = y
df.head()
```

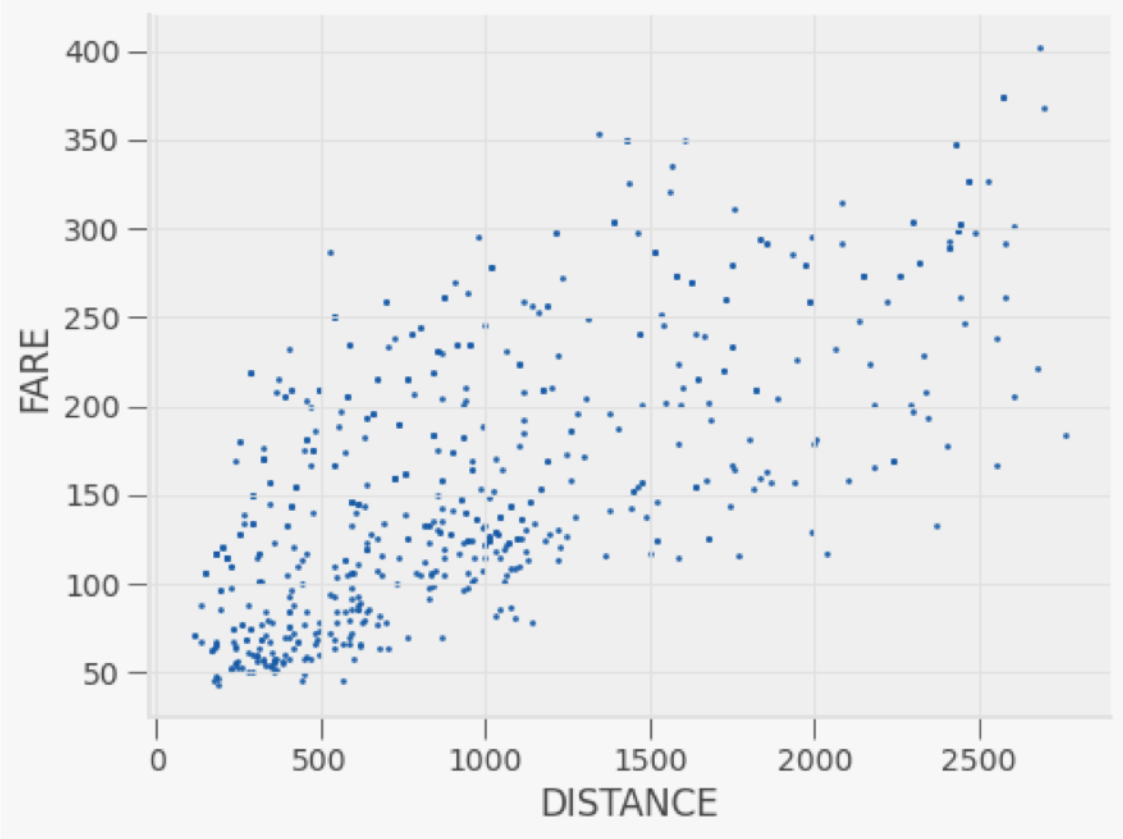
[5]:

	S_CODE	S_CITY	E_CODE	E_CITY	COUPON	NEW	\
0	*	Dallas/Fort Worth	TX	*	Amarillo	TX	1.00	3
1	*	Atlanta	GA	*	Baltimore/Wash Intl	MD	1.06	3
2	*	Boston	MA	*	Baltimore/Wash Intl	MD	1.06	3
3	ORD	Chicago	IL	*	Baltimore/Wash Intl	MD	1.06	3
4	MDW	Chicago	IL	*	Baltimore/Wash Intl	MD	1.06	3

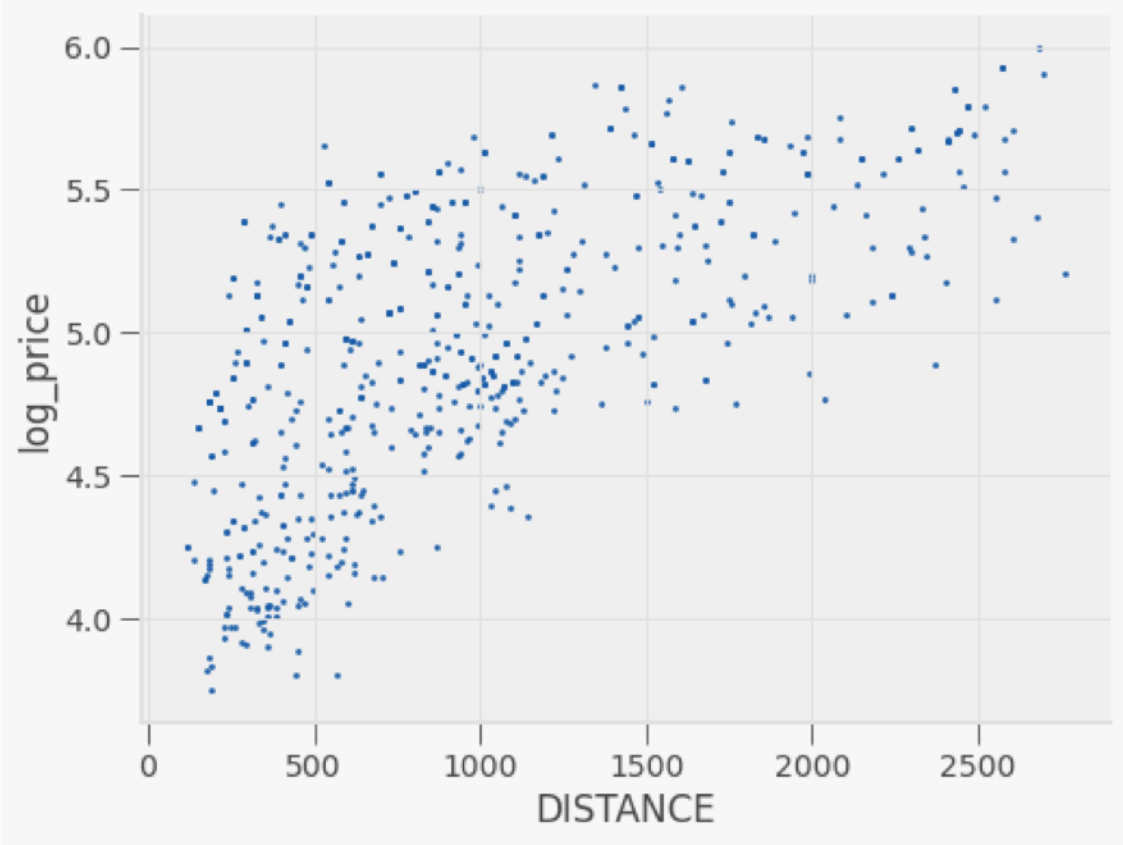
	VACATION	SW	HI	S_INCOME	E_INCOME	S_POP	E_POP	SLOT	\
0	No	Yes	5291.99	28637.0	21112.0	3036732	205711	Free	
1	No	No	5419.16	26993.0	29838.0	3532657	7145897	Free	
2	No	No	9185.28	30124.0	29838.0	5787293	7145897	Free	
3	No	Yes	2657.35	29260.0	29838.0	7830332	7145897	Controlled	
4	No	Yes	2657.35	29260.0	29838.0	7830332	7145897	Free	

	GATE	DISTANCE	PAX	FARE	log_price
0	Free	312	7864	64.11	4.160600
1	Free	576	8820	174.47	5.161753
2	Free	364	6452	207.76	5.336384
3	Free	612	25144	85.47	4.448165
4	Free	612	25144	85.47	4.448165

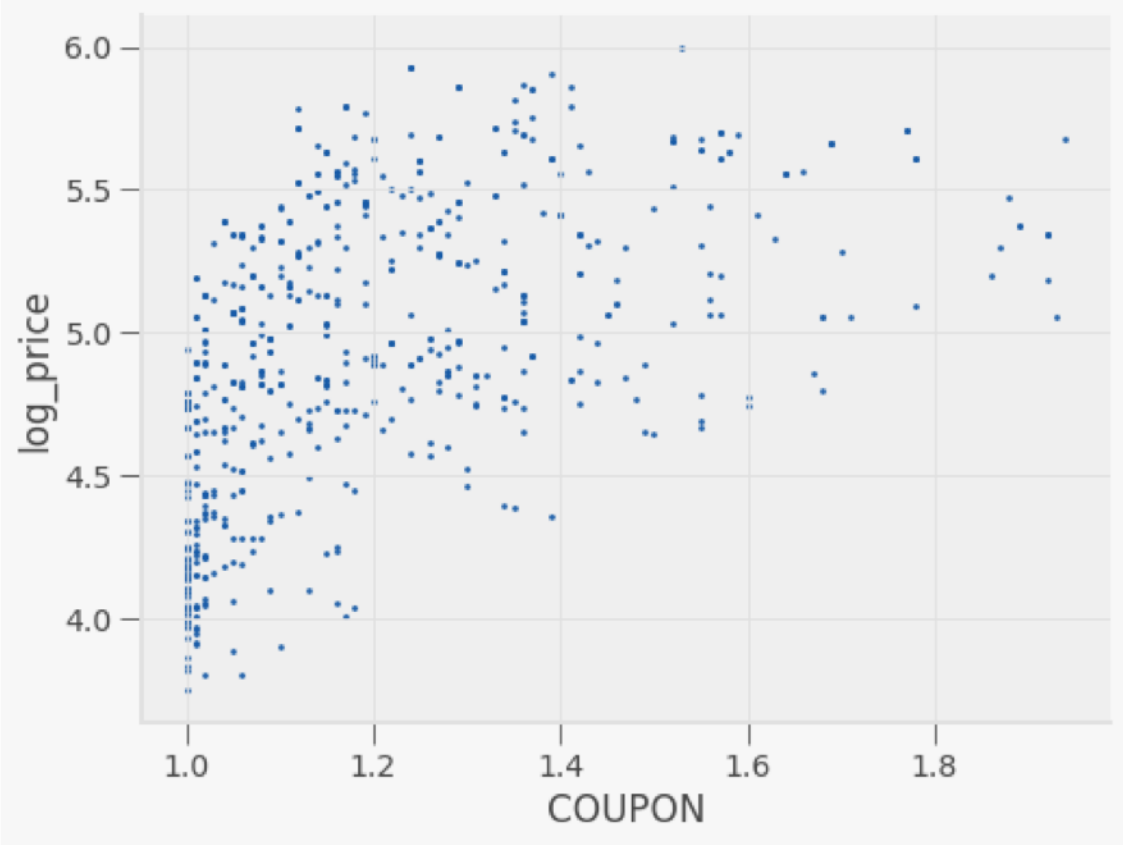
[6]:
```
def var_scatter(df, ax=None, var="DISTANCE"):
    if ax is None:
        _, ax = plt.subplots(figsize=(8, 6))
    df.plot.scatter(x=var , y="FARE",alpha=0.9, s=3, ax=ax)

    return ax

var_scatter(df);
plt.savefig("scatter1.png")
```



[7]:
```
def var_scatter(df, ax=None, var="DISTANCE"):
    if ax is None:
        _, ax = plt.subplots(figsize=(8, 6))
    df.plot.scatter(x=var , y="log_price",alpha=0.9, s=3, ax=ax)

    return ax

var_scatter(df);
plt.savefig("scatter1.png")
```



[8]:
```
def var_scatter(df, ax=None, var="COUPON"):
    if ax is None:
        _, ax = plt.subplots(figsize=(8, 6))
    df.plot.scatter(x=var , y="log_price",alpha=0.9, s=3, ax=ax)

    return ax

var_scatter(df);
plt.savefig("scatter1.png")
```



[9]:
```
df
```

[9]:

	S_CODE	S_CITY	E_CODE	E_CITY	COUPON	\
0	*	Dallas/Fort Worth	TX	* Amarillo	TX	1.00
1	*	Atlanta	GA	* Baltimore/Wash Intl	MD	1.06
2	*	Boston	MA	* Baltimore/Wash Intl	MD	1.06
3	ORD	Chicago	IL	* Baltimore/Wash Intl	MD	1.06
4	MDW	Chicago	IL	* Baltimore/Wash Intl	MD	1.06
..	…	…	…	…	…	
633	LGA	New York/Newark	NY	* West Palm Beach	FL	1.08
634	EWR	New York/Newark	NY	* West Palm Beach	FL	1.08
635	*	Philadelphia/Camden	PA	* West Palm Beach	FL	1.17
636	IAD	Washington	DC	* West Palm Beach	FL	1.28
637	DCA	Washington	DC	* West Palm Beach	FL	1.28

	NEW	VACATION	SW	HI	S_INCOME	E_INCOME	S_POP	E_POP	\
0	3	No	Yes	5291.99	28637.0	21112.0	3036732	205711	
1	3	No	No	5419.16	26993.0	29838.0	3532657	7145897	
2	3	No	No	9185.28	30124.0	29838.0	5787293	7145897	
3	3	No	Yes	2657.35	29260.0	29838.0	7830332	7145897	
4	3	No	Yes	2657.35	29260.0	29838.0	7830332	7145897	

```
..   …      …  …      …        …        …       …       …
633    3      Yes    No  2216.70   32991.0   37375.0  8621121   991717
634    0      Yes    No  2216.70   32991.0   37375.0  8621121   991717
635    3      Yes    No  6797.80   27994.0   37375.0  4948339   991717
636    3      Yes    No  5566.43   31981.0   37375.0  4549784   991717
637    3      Yes    No  5566.43   31981.0   37375.0  4549784   991717

           SLOT         GATE  DISTANCE    PAX    FARE  log_price
0          Free         Free       312   7864   64.11   4.160600
1          Free         Free       576   8820  174.47   5.161753
2          Free         Free       364   6452  207.76   5.336384
3    Controlled         Free       612  25144   85.47   4.448165
4          Free         Free       612  25144   85.47   4.448165
..          …           …       …    …     …       …
633  Controlled         Free      1030  34324  129.63   4.864684
634        Free  Constrained      1030  34324  129.63   4.864684
635        Free         Free       960   6016  124.87   4.827273
636        Free         Free       858   4877  129.62   4.864607
637  Controlled         Free       858   4877  129.62   4.864607

[638 rows x 19 columns]
```

There are VACATION, SW,SLOT, and GATE that are counted as the categorical data. Therefore, if we need to use these information in our regression model, we need to convert it to be the dummy variables.

[10]:
```
df=pd.get_dummies(df,columns=['VACATION', 'SW','SLOT','GATE'],drop_first=True)
```

[11]:
```
list(df)
```

[11]:
```
['S_CODE',
 'S_CITY',
 'E_CODE',
 'E_CITY',
 'COUPON',
 'NEW',
 'HI',
 'S_INCOME',
 'E_INCOME',
 'S_POP',
 'E_POP',
 'DISTANCE',
 'PAX',
 'FARE',
 'log_price',
 'VACATION_Yes',
 'SW_Yes',
```

```
 'SLOT_Free',
 'GATE_Free']
```

[12]:
```
#model 1:
```

[13]:
```
X=df[['COUPON',
 'DISTANCE',
 'VACATION_Yes']]
y=df[['log_price']]
X=X.to_numpy()
y=y.to_numpy()
```

[14]:
```
from sklearn.model_selection import KFold
kf = KFold(n_splits=5,shuffle=True)
```

[15]:
```
err_train = 0
err_test =0
for train,test in kf.split(X):
    lr=linear_model.LinearRegression()
    reg=lr.fit(X[train],y[train])
    y_pred_train =reg.predict(X[train])
    y_pred_test =reg.predict(X[test])
    e_train= y[train]-y_pred_train
    e_test = y[test]-y_pred_test
    err_train += np.sqrt(np.mean(e_train*e_train))     # compute rmse for the␣
 ↪estimation rmse test data
    err_test += np.sqrt(np.mean(e_test*e_test))
rmse_train_5cv_model1 = err_train/5                #average the rmse
rmse_test_5cv_model1 = err_test/5                 #average the rmse
print('RMSE on 5-fold CV on the train data: {}'.format(rmse_train_5cv_model1))
print('RMSE on 5-fold CV on the test data: {}'.format(rmse_test_5cv_model1))
```

```
RMSE on 5-fold CV on the train data: 0.35066051937292386
RMSE on 5-fold CV on the test data: 0.3521068477995167
```

2 Alternative way to calculate the RMSE

[16]:
```
err_train = 0
err_test =0
for train,test in kf.split(X):
    lr=linear_model.LinearRegression()
    reg=lr.fit(X[train],y[train])
    y_pred_train =reg.predict(X[train])
    y_pred_test =reg.predict(X[test])
    e_train= y[train]-y_pred_train
    e_test = y[test]-y_pred_test
```

```
    err_train += np.sqrt(metrics.mean_squared_error(y[train], reg.
 ↪predict(X[train])))
    err_test += np.sqrt(metrics.mean_squared_error(y[test], reg.
 ↪predict(X[test])))

rmse_train_5cv_model1 = err_train/5              #average the rmse
rmse_test_5cv_model1 = err_test/5               #average the rmse
print('RMSE (Model1:Linear) on 5-fold CV on the train data: {}'.
 ↪format(rmse_train_5cv_model1))
print('RMSE (Model1:Linear) on 5-fold CV on the test data: {}'.
 ↪format(rmse_test_5cv_model1))
```

```
RMSE (Model1:Linear) on 5-fold CV on the train data: 0.3507727349525686
RMSE (Model1:Linear) on 5-fold CV on the test data: 0.35118351783147095
```

[17]:
```
err_train = 0
err_test =0
for train,test in kf.split(X):
    lr=linear_model.LinearRegression()
    reg=lr.fit(X[train],y[train])
    y_pred_train =reg.predict(X[train])
    y_pred_test =reg.predict(X[test])
    e_train= y[train]-y_pred_train
    e_test = y[test]-y_pred_test
    err_train += metrics.mean_absolute_error(y[train], reg.predict(X[train]))
    err_test += metrics.mean_absolute_error(y[test], reg.predict(X[test]))
mae_train_5cv_model1 = err_train/5              #average the rmse
mae_test_5cv_model1 = err_test/5               #average the rmse
print('MAE (Model1:Linear) on 5-fold CV on the train data: {}'.
 ↪format(mae_train_5cv_model1))
print('MAE (Model1:Linear) on 5-fold CV on the test data: {}'.
 ↪format(mae_test_5cv_model1))
```

```
MAE (Model1:Linear) on 5-fold CV on the train data: 0.2899921832415116
MAE (Model1:Linear) on 5-fold CV on the test data: 0.2918824010247195
```

[18]:
```
# Summary of model1:
print('RMSE (Model1:Linear) on 5-fold CV on the train data: {}'.
 ↪format(rmse_train_5cv_model1))
print('RMSE (Model1:Linear) on 5-fold CV on the test data: {}'.
 ↪format(rmse_test_5cv_model1))
```

```
RMSE (Model1:Linear) on 5-fold CV on the train data: 0.3507727349525686
RMSE (Model1:Linear) on 5-fold CV on the test data: 0.35118351783147095
```

[19]:
```
print('MAE (Model1:Linear) on 5-fold CV on the train data: {}'.
 ↪format(mae_train_5cv_model1))
```

```
print('MAE (Model1:Linear) on 5-fold CV on the test data: {}'.
 ↪format(mae_test_5cv_model1))
```

```
MAE (Model1:Linear) on 5-fold CV on the train data: 0.2899921832415116
MAE (Model1:Linear) on 5-fold CV on the test data: 0.2918824010247195
```

3 Now, we conduct the model 2

[20]:
```
X=df[['COUPON',
 'DISTANCE',
 'VACATION_Yes','GATE_Free']]
y=df[['log_price']]
X=X.to_numpy()
y=y.to_numpy()
```

[21]:
```
err_train = 0
err_test =0
for train,test in kf.split(X):
    lr=linear_model.LinearRegression()
    reg=lr.fit(X[train],y[train])
    y_pred_train =reg.predict(X[train])
    y_pred_test =reg.predict(X[test])
    e_train= y[train]-y_pred_train
    e_test = y[test]-y_pred_test
    err_train += np.sqrt(metrics.mean_squared_error(y[train], reg.
 ↪predict(X[train])))
    err_test += np.sqrt(metrics.mean_squared_error(y[test], reg.
 ↪predict(X[test])))

rmse_train_5cv_model2 = err_train/5              #average the rmse
rmse_test_5cv_model2 = err_test/5               #average the rmse
print('RMSE (Model2:Linear) on 5-fold CV on the train data: {}'.
 ↪format(rmse_train_5cv_model2))
print('RMSE (Model2:Linear) on 5-fold CV on the test data: {}'.
 ↪format(rmse_test_5cv_model2))
```

```
RMSE (Model2:Linear) on 5-fold CV on the train data: 0.3276909739846823
RMSE (Model2:Linear) on 5-fold CV on the test data: 0.3301257349907664
```

[22]:
```
err_train = 0
err_test =0
for train,test in kf.split(X):
    lr=linear_model.LinearRegression()
    reg=lr.fit(X[train],y[train])
    y_pred_train =reg.predict(X[train])
    y_pred_test =reg.predict(X[test])
```

```
    e_train= y[train]-y_pred_train
    e_test = y[test]-y_pred_test
    err_train += metrics.mean_absolute_error(y[train], reg.predict(X[train]))
    err_test += metrics.mean_absolute_error(y[test], reg.predict(X[test]))
mae_train_5cv_model2 = err_train/5              #average the rmse
mae_test_5cv_model2 = err_test/5               #average the rmse
print('MAE (Model2:Linear) on 5-fold CV on the train data: {}'.
 ↪format(mae_train_5cv_model2))
print('MAE (Model2:Linear) on 5-fold CV on the test data: {}'.
 ↪format(mae_test_5cv_model2))
```

```
MAE (Model2:Linear) on 5-fold CV on the train data: 0.2743965939830425
MAE (Model2:Linear) on 5-fold CV on the test data: 0.27710590652660716
```

[23]:
```
print('MAE (Model1:Linear) on 5-fold CV on the train data: {}'.
 ↪format(mae_train_5cv_model1))
print('MAE (Model1:Linear) on 5-fold CV on the test data: {}'.
 ↪format(mae_test_5cv_model1))
```

```
MAE (Model1:Linear) on 5-fold CV on the train data: 0.2899921832415116
MAE (Model1:Linear) on 5-fold CV on the test data: 0.2918824010247195
```

[24]:
```
print('MAE (Model2:Linear) on 5-fold CV on the train data: {}'.
 ↪format(mae_train_5cv_model2))
print('MAE (Model2:Linear) on 5-fold CV on the test data: {}'.
 ↪format(mae_test_5cv_model2))
```

```
MAE (Model2:Linear) on 5-fold CV on the train data: 0.2743965939830425
MAE (Model2:Linear) on 5-fold CV on the test data: 0.27710590652660716
```

4 Last, we conduct the model with all information

[25]:
```
list(df)
```

[25]:
```
['S_CODE',
 'S_CITY',
 'E_CODE',
 'E_CITY',
 'COUPON',
 'NEW',
 'HI',
 'S_INCOME',
 'E_INCOME',
 'S_POP',
 'E_POP',
 'DISTANCE',
```

```
 'PAX',
 'FARE',
 'log_price',
 'VACATION_Yes',
 'SW_Yes',
 'SLOT_Free',
 'GATE_Free']
```

[28]:
```
X=df[['COUPON',
 'NEW',
 'HI',
 'S_INCOME',
 'E_INCOME',
 'S_POP',
 'E_POP',
 'DISTANCE',
 'PAX',
 'FARE',
 'VACATION_Yes',
 'SW_Yes',
 'SLOT_Free',
 'GATE_Free']]
y=df[['log_price']]
X=X.to_numpy()
y=y.to_numpy()
```

[29]:
```
err_train = 0
err_test =0
for train,test in kf.split(X):
    lr=linear_model.LinearRegression()
    reg=lr.fit(X[train],y[train])
    y_pred_train =reg.predict(X[train])
    y_pred_test =reg.predict(X[test])
    e_train= y[train]-y_pred_train
    e_test = y[test]-y_pred_test
    err_train += np.sqrt(metrics.mean_squared_error(y[train], reg.
 ↪predict(X[train])))
    err_test += np.sqrt(metrics.mean_squared_error(y[test], reg.
 ↪predict(X[test])))

rmse_train_5cv_model3 = err_train/5              #average the rmse
rmse_test_5cv_model3 = err_test/5               #average the rmse
print('RMSE (Model3:Linear) on 5-fold CV on the train data: {}'.
 ↪format(rmse_train_5cv_model3))
print('RMSE (Model3:Linear) on 5-fold CV on the test data: {}'.
 ↪format(rmse_test_5cv_model3))
```

```
RMSE (Model3:Linear) on 5-fold CV on the train data: 0.10912336441067641
RMSE (Model3:Linear) on 5-fold CV on the test data: 0.11455464358336528
```

[ ]:
```
Based on the RMSE, the model with all information performs the best.
```