ML Pipeline

December 2, 2022

0.0.1 Package Used

[14]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder,␣
↪OrdinalEncoder, MinMaxScaler
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

### Model
import xgboost
from sklearn.model_selection import ParameterGrid
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.ensemble import RandomForestRegressor
from  sklearn.linear_model import Ridge
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
```

[15]:
```
Airbnb = pd.read_csv("../Data/EDA_data.csv")
X = Airbnb.loc[:,Airbnb.columns!="price"]
y = Airbnb["price"]
```

0.1 Spliting

[16]:
```
# Split
X_train, X_other, y_train, y_other = train_test_split(X,y,train_size = 0.
 ↪6,random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_other,y_other,train_size = 0.
 ↪5,random_state=42)
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
(29330, 11)
(9777, 11)
(9777, 11)
```

[17]:
```
# ## Train
# y_train.plot.hist(log=True, bins = np.logspace(np.log10(1),np.log10(np.
 ↪max(y_train)),50))
# plt.semilogy()
# plt.semilogx()
# plt.xlabel('Airbnb price $/night')
# plt.ylabel('Count')
# plt.title('Train set Price Distribution')
# plt.show()
#
# ## Validation
# y_val.plot.hist(log=True, bins = np.logspace(np.log10(1),np.log10(np.
 ↪max(y_train)),50))
# plt.semilogy()
# plt.semilogx()
# plt.xlabel('Airbnb price $/night')
# plt.ylabel('Count')
# plt.title('Validation set Price Distribution')
# plt.show()
#
# ## Test
# y_test.plot.hist(log=True, bins = np.logspace(np.log10(1),np.log10(np.
 ↪max(y_train)),50))
# plt.semilogy()
# plt.semilogx()
# plt.xlabel('Airbnb price $/night')
# plt.ylabel('Count')
# plt.title('Test set Price Distribution')
# plt.show()
```

0.2 Preprocessing

Check data type

[18]:
```
#X.dtypes
```

Check Missing Data

[19]:
```
print('fraction of missing values in features:')
print("\n")
perc_missing_per_ftr = X.isnull().sum(axis=0)/X.shape[0]
print(perc_missing_per_ftr[perc_missing_per_ftr>0])
```

```
fraction of missing values in features:


reviews_per_month                          0.205609
gap_between_last_review_and_end_of_2019    0.205609
dtype: float64
```

There are two features in the feature matric contain missing values and they are `reviews_per_month` and `gap_between_last_review_and_end_of_2019`. Their corresponding missing proportion are **20.56%**, and **20.56%**. All two features are **continuous** variable. In later analysis, we will use **reduced-features model** to deal with missingness.

Preprocess

[20]:
```
ordinal_ftrs= ["room_type"]
ordinal_cats = [["Shared room", "Private room", "Entire home/apt"]]
onehot_ftrs = ["neighbourhood_group", "neighbourhood"]
minmax_ftrs = ["availability_365"]
std_ftrs = ["minimum_nights", "number_of_reviews", "longitude", "latitude",
            "gap_between_last_review_and_end_of_2019","reviews_per_month",␣
 ↪"calculated_host_listings_count"]
# collect all the encoders
preprocessor = ColumnTransformer(
    transformers=[
        ('ord', OrdinalEncoder(categories = ordinal_cats), ordinal_ftrs),
        ('onehot', OneHotEncoder(sparse=False,handle_unknown='ignore'),␣
 ↪onehot_ftrs),
        ('minmax', MinMaxScaler(), minmax_ftrs),
        ('std', StandardScaler(), std_ftrs)])

# fit_transform the training set
X_prep = preprocessor.fit_transform(X_train)
# collect feature names
feature_names = preprocessor.get_feature_names_out()

df_train = pd.DataFrame(data=X_prep,columns=feature_names)
print(df_train.shape)

# transform the CV
df_val = preprocessor.transform(X_val)
df_val = pd.DataFrame(data=df_val,columns = feature_names)
print(df_val.shape)
```

```
# transform the test
df_test = preprocessor.transform(X_test)
df_test = pd.DataFrame(data=df_test,columns = feature_names)
print(df_test.shape)
```

```
(29330, 230)
(9777, 230)
(9777, 230)
```

Check strong correlated features

[21]:
```
## Check any strong correlated features
corr = df_train.corr().round(2)
n = 0
for i in range(230):
    if sum(corr.iloc[i] < -0.8)>0:
        n+=1
        print(i)
    if sum(corr.iloc[i] >  0.8)>1:
        n+=1
        print(i)
print("Any absolute correlation score larger than 0.8: ",str(n))
```

```
Any absolute correlation score larger than 0.8:  0
```

0.2.1 Modeling

Baseline Score (RMSE)

[22]:
```
RMSE_base = []
for i in range(10):
    X_train, X_other, y_train, y_other = train_test_split(X,y,train_size = 0.
 ↪6,random_state=42+i)
    X_val, X_test, y_val, y_test = train_test_split(X_other,y_other,train_size␣
 ↪= 0.5,random_state=42+i)
    baseline_pred = [np.mean(y_train) for i in range(len(y_test))]
    RMSE_base.append(np.sqrt(mean_squared_error(y_test,baseline_pred)))
print("Baseline RMSE Mean: ",str(np.mean(RMSE_base)))
print("Baseline RMSE Std: ",str(np.std(RMSE_base)))
```

```
Baseline RMSE Mean:  234.44447309264723
Baseline RMSE Std:  28.439869512106657
```

[23]:
```
RMSE_base
```

[23]:
```
[191.26127725683043,
 265.00762405674374,
 202.3112267176499,
 197.04268354373804,
 234.2071358439469,
```

```
 267.4048023112737,
 227.15109417514387,
 256.16214843370005,
 269.75096181968837,
 234.1457767677575]
```

Check Pattern

[24]:
```python
print('data dimensions:',df_train.shape)
perc_missing_per_ftr = df_train.isnull().sum(axis=0)/df_train.shape[0]
print('fraction of missing values in features:')
print(perc_missing_per_ftr[perc_missing_per_ftr > 0])
frac_missing = sum(df_train.isnull().sum(axis=1)!=0)/df_train.shape[0]
print('fraction of points with missing values:',frac_missing)
print("\n\nUnique Patterns:")
mask =␣
 ↪df_test[['std__gap_between_last_review_and_end_of_2019','std__reviews_per_month']].
 ↪isnull()
unique_rows, counts = np.unique(mask, axis=0,return_counts=True)
print(unique_rows.shape) # 6 patterns, we will train 6 models
for i in range(len(counts)):
    print(unique_rows[i],counts[i])
```

```
data dimensions: (29330, 230)
fraction of missing values in features:
std__gap_between_last_review_and_end_of_2019    0.205421
std__reviews_per_month                          0.205421
dtype: float64
fraction of points with missing values: 0.20542107057620185


Unique Patterns:
(2, 2)
[False False] 7752
[ True  True] 2025
```

ML Model

[25]:
```python
def Random_forest_model(param_grid, final_models, X_train, y_train, X_val,␣
 ↪y_val, X_test, y_test):


    ## Initial list
    train_score = np.zeros(len(ParameterGrid(param_grid)))
    val_score = np.zeros(len(ParameterGrid(param_grid)))
    models = []

    ## CV
    for p in range(len(ParameterGrid(param_grid))):
```

```
        params = ParameterGrid(param_grid)[p]
        clf = RandomForestRegressor(**params,random_state = 42*p,n_jobs=-1)

        # loop through all posible model
        clf.fit(X_train,y_train)
        models.append(clf)

        # calculate train and validation accuracy scores
        ## Train
        y_train_pred = clf.predict(X_train)
        train_score[p] = np.sqrt(mean_squared_error(y_train,y_train_pred))
        ## Validation
        y_val_pred = clf.predict(X_val)
        val_score[p] = np.sqrt(mean_squared_error(y_val,y_val_pred))

    #print("-----------------------------------------------------------")
    #print('best model parameters:',ParameterGrid(param_grid)[np.
 ↪argmin(val_score)])
    print('corresponding validation score:',np.min(val_score))

    ## Model with best validation
    final_models.append(models[np.argmin(val_score)])

    ## Test
    y_test_pred = final_models[-1].predict(X_test)

    # Output test prediction
    return(y_test_pred)

def Ridge_model(param_grid, final_models, X_train, y_train, X_val, y_val,␣
 ↪X_test, y_test):


    ## Initial list
    train_score = np.zeros(len(ParameterGrid(param_grid)))
    val_score = np.zeros(len(ParameterGrid(param_grid)))
    models = []

    ## CV
    for p in range(len(ParameterGrid(param_grid))):
        params = ParameterGrid(param_grid)[p]
        clf = Ridge(**params, random_state = 42, max_iter=100000000)

        # loop through all posible model
        clf.fit(X_train,y_train)
        models.append(clf)
```

```
        # calculate train and validation accuracy scores
        ## Train
        y_train_pred = clf.predict(X_train)
        train_score[p] = np.sqrt(mean_squared_error(y_train,y_train_pred))
        ## Validation
        y_val_pred = clf.predict(X_val)
        val_score[p] = np.sqrt(mean_squared_error(y_val,y_val_pred))

    #print("------------------------------------------------------")
    #print('best model parameters:',ParameterGrid(param_grid)[np.
 ↪argmin(val_score)])
    print('corresponding validation score:',np.min(val_score))

    ## Model with best validation
    final_models.append(models[np.argmin(val_score)])

    ## Test
    y_test_pred = final_models[-1].predict(X_test)

    # Output test prediction
    return(y_test_pred)

def SVR_model(param_grid, final_models, X_train, y_train, X_val, y_val, X_test,␣
 ↪y_test):


    ## Initial list
    train_score = np.zeros(len(ParameterGrid(param_grid)))
    val_score = np.zeros(len(ParameterGrid(param_grid)))
    models = []

    ## CV
    for p in range(len(ParameterGrid(param_grid))):
        params = ParameterGrid(param_grid)[p]
        clf = SVR(**params)

        # loop through all posible model
        clf.fit(X_train,y_train)
        models.append(clf)
        # calculate train and validation accuracy scores
        ## Train
        y_train_pred = clf.predict(X_train)
        train_score[p] = np.sqrt(mean_squared_error(y_train,y_train_pred))
        ## Validation
        y_val_pred = clf.predict(X_val)
        val_score[p] = np.sqrt(mean_squared_error(y_val,y_val_pred))
```

```
    #print("-----------------------------------------------------------")
    #print('best model parameters:',ParameterGrid(param_grid)[np.
 ↪argmin(val_score)])
    print('corresponding validation score:',np.min(val_score))

    ## Model with best validation
    final_models.append(models[np.argmin(val_score)])

    ## Test
    y_test_pred = final_models[-1].predict(X_test)

    # Output test prediction
    return(y_test_pred)

def KNN_model(param_grid, final_models, X_train, y_train, X_val, y_val, X_test,␣
 ↪y_test):
    ## Initial list
    train_score = np.zeros(len(ParameterGrid(param_grid)))
    val_score = np.zeros(len(ParameterGrid(param_grid)))
    models = []

    ## CV
    for p in range(len(ParameterGrid(param_grid))):
        params = ParameterGrid(param_grid)[p]
        clf = KNeighborsRegressor(**params, n_jobs =-1)

        # loop through all posible model
        clf.fit(X_train,y_train)
        models.append(clf)
        # calculate train and validation accuracy scores
        ## Train
        y_train_pred = clf.predict(X_train)
        train_score[p] = np.sqrt(mean_squared_error(y_train,y_train_pred))
        ## Validation
        y_val_pred = clf.predict(X_val)
        val_score[p] = np.sqrt(mean_squared_error(y_val,y_val_pred))

    #print("-----------------------------------------------------------")
    #print('best model parameters:',ParameterGrid(param_grid)[np.
 ↪argmin(val_score)])
    print('corresponding validation score:',np.min(val_score))

    ## Model with best validation
    final_models.append(models[np.argmin(val_score)])

    ## Test
    y_test_pred = final_models[-1].predict(X_test)
```

```
    # Output test prediction
    return(y_test_pred)
```

Pattern submodel approach

[26]:
```
def reduced_feature(df_train, y_train, df_val, y_val, df_test, y_test,␣
 ↪param_grid, ML):
    if ML not in ["RF","Ridge","SVR","KNN"]:
        raise ValueError('Please select a valiad ML method')
    mask = df_test.isnull()
    unique_rows = np.array(np.unique(mask, axis=0))
    all_y_test_pred = pd.DataFrame()
    print('there are', len(unique_rows), 'unique missing value patterns.')

    for i in range(len(unique_rows)):
        print('working on unique pattern', i)
        ## generate X_test subset that matches the unique pattern i
        sub_X_test = pd.DataFrame()
        sub_y_test = pd.Series(dtype=float)
        for j in range(len(mask)):
            row_mask = np.array(mask.iloc[j])
            if np.array_equal(row_mask, unique_rows[i]):
                sub_X_test = sub_X_test.append(df_test.iloc[j])
                sub_y_test = sub_y_test.append(y_test.iloc[[j]])

        sub_X_test = sub_X_test[df_test.columns[~unique_rows[i]]]

        ## choose the according reduced features for subgroups
        sub_X_train = pd.DataFrame()
        sub_y_train = pd.DataFrame()
        sub_X_val = pd.DataFrame()
        sub_y_val = pd.DataFrame()

        # 1.cut the feature columns that have nans in the according sub_X_test
        sub_X_train = df_train[df_train.columns[~unique_rows[i]]]
        sub_X_val = df_val[df_val.columns[~unique_rows[i]]]

        # 2.cut the rows in the sub_X_train and sub_X_CV that have any nans
        sub_X_train = sub_X_train.dropna()
        sub_X_val = sub_X_val.dropna()

        # 3.cut the sub_Y_train and sub_y_CV accordingly
        sub_y_train = y_train.iloc[sub_X_train.index]
        sub_y_val = y_val.iloc[sub_X_val.index]
        ############################ model ##############################

        final_models = []
```

```
        if ML == "RF":
            sub_y_test_pred = Random_forest_model(param_grid, final_models,
                                                  sub_X_train, sub_y_train,
                                                  sub_X_val, sub_y_val,
                                                  sub_X_test, sub_y_test)
        elif ML =="Ridge":
            sub_y_test_pred = Ridge_model(param_grid, final_models,
                                                  sub_X_train, sub_y_train,
                                                  sub_X_val, sub_y_val,
                                                  sub_X_test, sub_y_test)
        elif ML == "SVR":
            sub_y_test_pred = SVR_model(param_grid, final_models,
                                                  sub_X_train, sub_y_train,
                                                  sub_X_val, sub_y_val,
                                                  sub_X_test, sub_y_test)
        elif ML == "KNN":
            sub_y_test_pred = KNN_model(param_grid, final_models,
                                                  sub_X_train, sub_y_train,
                                                  sub_X_val, sub_y_val,
                                                  sub_X_test, sub_y_test)

        sub_y_test_pred = pd.
↪DataFrame(sub_y_test_pred,columns=['sub_y_test_pred'], index=sub_y_test.
↪index)
        sub_test_score = np.sqrt(mean_squared_error(sub_y_test,sub_y_test_pred))
        print('   RMSE:',sub_test_score)
        all_y_test_pred = all_y_test_pred.append(sub_y_test_pred)
    all_y_test_pred = all_y_test_pred.sort_index()
    y_test = y_test.sort_index()

    # get global RMSE
    total_RMSE = np.sqrt(mean_squared_error(y_test,all_y_test_pred))
    total_R2 =  r2_score(y_test,all_y_test_pred)
    print("-----------------------------------")
    print("Total RMSE: ",total_RMSE)
    print("total_R2: ",total_R2)
    print("-----------------------------------")
    return(total_RMSE, total_R2)
```

0.2.2 Parameter tuning

[14]:
```
def ML_pipe(X,y,preprocessor,param_grid,ML,iteration = 3):
    test_RMSE = []
    test_R2 = []
    for i in range(iteration):
        print("  iteration",i,":")
```

```
        X_train, X_other, y_train, y_other = train_test_split(X,y,train_size =␣
↪0.6,random_state=42*i)
        X_val, X_test, y_val, y_test =␣
↪train_test_split(X_other,y_other,train_size = 0.5,random_state=42*i)

        # fit_transform the training set
        X_prep = preprocessor.fit_transform(X_train)
        # collect feature names
        feature_names = preprocessor.get_feature_names_out()

        df_train = pd.DataFrame(data=X_prep,columns=feature_names)
        print(df_train.shape)

        # transform the CV
        df_val = preprocessor.transform(X_val)
        df_val = pd.DataFrame(data=df_val,columns = feature_names)
        print(df_val.shape)

        # transform the test
        df_test = preprocessor.transform(X_test)
        df_test = pd.DataFrame(data=df_test,columns = feature_names)
        print(df_test.shape)
        test_score = reduced_feature(df_train, y_train, df_val, y_val, df_test,␣
↪y_test, param_grid, ML)
        test_RMSE.append(test_score[0])
        test_R2.append(test_score[1])
    print(ML,"Test_RMSE Mean:", np.mean(test_RMSE))
    print(ML,"Test_RMSE Std:", np.std(test_RMSE))
    print(ML,"Test_R2 Mean:", np.mean(test_R2))
    print(ML,"Test_R2 Std:", np.std(test_R2))
    return(test_RMSE,test_R2)
```

Random Forest

[15]:
```
param_grid = {'max_depth': [1, 3, 10, 30, 100],
              'max_features': [0.25, 0.5,0.75,1.0]}
RF = ML_pipe(X,y,preprocessor,param_grid,"RF",iteration = 10)
```

```
  iteration 0 :
(29330, 230)
(9777, 230)
(9777, 230)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 125.5824058206506
   RMSE: 228.07498352044706
working on unique pattern 1
corresponding validation score: 174.84379519718897
```

```
   RMSE: 414.13771970693386
-----------------------------------
Total RMSE:  276.1410661365781
total_R2:  0.10881600908470446
-----------------------------------
  iteration 1 :
(29330, 230)
(9777, 230)
(9777, 230)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 159.71290453881483
   RMSE: 156.93369441171083
working on unique pattern 1
corresponding validation score: 192.72639020127997
   RMSE: 242.4427894374151
-----------------------------------
Total RMSE:  178.04874703455653
total_R2:  0.13283523702985978
-----------------------------------
  iteration 2 :
(29330, 232)
(9777, 232)
(9777, 232)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 186.84289759130377
   RMSE: 196.97081143145212
working on unique pattern 1
corresponding validation score: 200.25948910205713
   RMSE: 318.76851506337977
-----------------------------------
Total RMSE:  227.47218598619125
total_R2:  0.14627331235958763
-----------------------------------
  iteration 3 :
(29330, 231)
(9777, 231)
(9777, 231)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 198.1462194374287
   RMSE: 149.30383601658113
working on unique pattern 1
corresponding validation score: 209.2572403199079
   RMSE: 337.2063949742794
-----------------------------------
Total RMSE:  203.3862396624895
```

```
total_R2:  0.105190117569405
----------------------------------
  iteration 4 :
(29330, 229)
(9777, 229)
(9777, 229)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 159.35103722538486
   RMSE: 171.30843301758676
working on unique pattern 1
corresponding validation score: 193.64769227262178
   RMSE: 378.87642590899816
----------------------------------
Total RMSE:  229.1741617562278
total_R2:  0.10412702866747559
----------------------------------
  iteration 5 :
(29330, 231)
(9777, 231)
(9777, 231)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 156.2542227294556
   RMSE: 129.413036708635
working on unique pattern 1
corresponding validation score: 179.22061157604503
   RMSE: 296.6262649827306
----------------------------------
Total RMSE:  177.42855345044293
total_R2:  0.21625297940755395
----------------------------------
  iteration 6 :
(29330, 234)
(9777, 234)
(9777, 234)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 159.97649509687167
   RMSE: 172.3867256273891
working on unique pattern 1
corresponding validation score: 215.81886736337404
   RMSE: 325.545184940425
----------------------------------
Total RMSE:  213.09896980933527
total_R2:  0.20971199073850422
----------------------------------
  iteration 7 :
```

```
(29330, 232)
(9777, 232)
(9777, 232)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 162.61986874382862
   RMSE: 201.51439205250855
working on unique pattern 1
corresponding validation score: 212.04935448612287
   RMSE: 280.2397509945383
-----------------------------------
Total RMSE:  219.67634682049803
total_R2:  0.12463579337204234
-----------------------------------
  iteration 8 :
(29330, 231)
(9777, 231)
(9777, 231)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 125.99241412574867
   RMSE: 221.19470713453367
working on unique pattern 1
corresponding validation score: 179.28769592618792
   RMSE: 355.91913269218145
-----------------------------------
Total RMSE:  255.08943348565745
total_R2:  0.12279836083592166
-----------------------------------
  iteration 9 :
(29330, 232)
(9777, 232)
(9777, 232)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 215.00044492754992
   RMSE: 131.85782563931411
working on unique pattern 1
corresponding validation score: 227.54455441289804
   RMSE: 362.04803018734754
-----------------------------------
Total RMSE:  202.86752651755089
total_R2:  0.21767109778212845
-----------------------------------
RF Test_RMSE Mean: 218.23832306595278
RF Test_RMSE Std: 29.464478166739504
RF Test_R2 Mean: 0.14883119268471828
RF Test_R2 Std: 0.04474467688749921
```

Ridge

[16]:
```
param_grid = {"alpha": np.logspace(-10,0,10)}
Ridge = ML_pipe(X,y,preprocessor,param_grid,"Ridge",iteration = 10)
```

```
  iteration 0 :
(29330, 230)
(9777, 230)
(9777, 230)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 132.15896992640438
   RMSE: 232.12184356218322
working on unique pattern 1
corresponding validation score: 180.7878022479261
   RMSE: 424.99104322106933
-----------------------------------
Total RMSE:  282.1093574391719
total_R2:  0.06987703946927526
-----------------------------------
  iteration 1 :
(29330, 230)
(9777, 230)
(9777, 230)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 163.834596041539
   RMSE: 162.55882109800584
working on unique pattern 1
corresponding validation score: 199.80019773581628
   RMSE: 218.41448129868596
-----------------------------------
Total RMSE:  175.59258780062044
total_R2:  0.1565950664600102
-----------------------------------
  iteration 2 :
(29330, 232)
(9777, 232)
(9777, 232)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 191.26546092282118
   RMSE: 200.62920500730507
working on unique pattern 1
corresponding validation score: 204.58553885849437
   RMSE: 324.96081774151315
-----------------------------------
Total RMSE:  231.77558772402924
total_R2:  0.11366553432839521
```

```
-----------------------------------
  iteration 3 :
(29330, 231)
(9777, 231)
(9777, 231)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 217.83493612766497
   RMSE: 130.29071869213396
working on unique pattern 1
corresponding validation score: 221.87409504661375
   RMSE: 356.1681121527491
-----------------------------------
Total RMSE:  199.75602553905256
total_R2:  0.13684773393689975
-----------------------------------
  iteration 4 :
(29330, 229)
(9777, 229)
(9777, 229)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 172.79766158065414
   RMSE: 164.0072626123222
working on unique pattern 1
corresponding validation score: 208.00832290478556
   RMSE: 390.6833251097796
-----------------------------------
Total RMSE:  228.93929810479167
total_R2:  0.10596231568115899
-----------------------------------
  iteration 5 :
(29330, 231)
(9777, 231)
(9777, 231)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 155.85099497394145
   RMSE: 137.28561846999975
working on unique pattern 1
corresponding validation score: 181.83369848138227
   RMSE: 307.68186905648633
-----------------------------------
Total RMSE:  185.81706973872033
total_R2:  0.14039270287540173
-----------------------------------
  iteration 6 :
(29330, 234)
```

```
(9777, 234)
(9777, 234)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 161.5081692857809
   RMSE: 182.31707924423128
working on unique pattern 1
corresponding validation score: 219.4154986129148
   RMSE: 345.923062992153
-----------------------------------
Total RMSE:  225.88598797545222
total_R2:  0.11202390185573896
-----------------------------------
  iteration 7 :
(29330, 232)
(9777, 232)
(9777, 232)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 167.38237354132946
   RMSE: 204.73271353260435
working on unique pattern 1
corresponding validation score: 219.11188755519436
   RMSE: 275.17254844810407
-----------------------------------
Total RMSE:  220.7572621095359
total_R2:  0.11600015791454732
-----------------------------------
  iteration 8 :
(29330, 231)
(9777, 231)
(9777, 231)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 129.4756008656426
   RMSE: 233.11431923413494
working on unique pattern 1
corresponding validation score: 181.33004279551028
   RMSE: 347.3157106660889
-----------------------------------
Total RMSE:  260.96980838711727
total_R2:  0.08188934126749514
-----------------------------------
  iteration 9 :
(29330, 232)
(9777, 232)
(9777, 232)
there are 2 unique missing value patterns.
```

```
working on unique pattern 0
corresponding validation score: 214.095275459817
   RMSE: 142.69242921241823
working on unique pattern 1
corresponding validation score: 227.97634387089062
   RMSE: 377.38071577354333
-----------------------------------
Total RMSE:  214.19201155762033
total_R2:  0.12789084430695696
-----------------------------------
Ridge Test_RMSE Mean: 222.5794996376112
Ridge Test_RMSE Std: 30.50802406809941
Ridge Test_R2 Mean: 0.11611446380958794
Ridge Test_R2 Std: 0.024963435788524422
```

SVR

[ ]:
```
param_grid = {'gamma': [1e-3, 1e-1, 1e1, 1e3, 1e5],
              'C': [1e-1, 1e0, 1e1]}
SVR = ML_pipe(X,y,preprocessor,param_grid,"SVR",iteration = 10)
```

```
  iteration 0 :
(29330, 230)
(9777, 230)
(9777, 230)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 134.19123873755342
   RMSE: 232.45152367719334
working on unique pattern 1
corresponding validation score: 183.09289372420156
   RMSE: 430.4157134222729
-----------------------------------
Total RMSE:  283.98757881509124
total_R2:  0.057450707111834864
-----------------------------------
  iteration 1 :
(29330, 230)
(9777, 230)
(9777, 230)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 164.09912012849784
   RMSE: 162.97627491647216
working on unique pattern 1
corresponding validation score: 200.5051660265449
   RMSE: 228.34358711604264
-----------------------------------
Total RMSE:  178.49164102811505
```

```
total_R2:  0.12851574873128502
-----------------------------------
  iteration 2 :
(29330, 232)
(9777, 232)
(9777, 232)
there are 2 unique missing value patterns.
working on unique pattern 0
corresponding validation score: 192.30065308810276
   RMSE: 201.38933411583787
working on unique pattern 1
```

KNN

[ ]:
```
param_grid = {'n_neighbors': [1,3,10,30,100],
               "weights" : ["distance","uniform"]}
KNN = ML_pipe(X,y,preprocessor,param_grid,"KNN",iteration = 10)
```

XGboost

[ ]:
```
import warnings
warnings.filterwarnings("ignore")

nr_states = 10
test_RMSE = np.zeros(nr_states)
test_R2   = np.zeros(nr_states)
final_models = []
test_y_sets = []
test_X_sets = []

for i in range(nr_states):
    print('randoms state '+str(i+1))
    # split function
    X_train, X_other, y_train, y_other = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=0+i)
    X_val, X_test, y_val, y_test = train_test_split(X_other, y_other,␣
 ↪test_size=0.5, random_state=0+i)

    # preprocess the sets
    X_train_prep = preprocessor.fit_transform(X_train)
    X_val_prep = preprocessor.transform(X_val)
    X_test_prep = preprocessor.transform(X_test)

    ## Creating train, val, and test dataframe
    feature_names = preprocessor.get_feature_names_out()
    df_train = pd.DataFrame(data=X_train_prep,columns=feature_names)
    df_val = pd.DataFrame(data=X_val_prep,columns = feature_names)
    df_test = pd.DataFrame(data=X_test_prep,columns = feature_names)
```

```
    ## Drop onehot__sex_Male
    #df_train= df_train.drop('onehot__sex_Male',axis = 1)
    #df_val= df_val.drop('onehot__sex_Male',axis = 1)
    #df_test= df_test.drop('onehot__sex_Male',axis = 1)

    ## Parameter
    param_grid = {#"learning_rate": [0.03],
                  "n_estimators": [10000],
                  "seed": [0],
                  #"reg_alpha": [0e0, 1e-2, 1e-1, 1e0, 1e1, 1e2],
                  "reg_lambda": [0e0, 1e-2, 1e-1, 1e0, 1e1, 1e2],
                  "missing": [np.nan],
                  "max_depth": [1,3,10,30,100],
                  #"colsample_bytree": [0.9],
                  #"subsample": [0.66]
                  }

    val_score = np.zeros(len(ParameterGrid(param_grid)))
    models = []

    for p in range(len(ParameterGrid(param_grid))):
        params = ParameterGrid(param_grid)[p]
        #print('   ',params)
        clf = xgboost.XGBRegressor(**params, n_jobs = -1, verbosity = 0)
        clf.fit(df_train,y_train,early_stopping_rounds=50,eval_set=[(df_val,␣
↪y_val)], verbose=False)
        models.append(clf)
        y_val_pred = clf.predict(df_val)

        val_score[p] = np.sqrt(mean_squared_error(y_val,y_val_pred))           ␣
↪        #sum(y_val_pred==y_val)/len(y_val)

    # print out model parameters that maximize validation accuracy
    print('best model parameters:',ParameterGrid(param_grid)[np.
↪argmin(val_score)])
    print('corresponding validation score:',np.min(val_score))
    # collect and save the best model
    final_models.append(models[np.argmin(val_score)])
    # calculate and save the test score
    y_test_pred = final_models[-1].predict(df_test)
    test_RMSE[i] = np.sqrt(mean_squared_error(y_test,y_test_pred))
    test_R2[i] = r2_score(y_test,y_test_pred)
    print('test RMSE score:',test_RMSE[i])
    print('test R2 score:',  test_R2[i])
    test_y_sets.append(y_test)
    test_X_sets.append(df_test)
```

```
print("------------------------------------------------")
print("mean and standard deviation of test accuracy score")
print("------------------------------------------------\n")
print("Mean:",str(np.mean(test_RMSE)))
print("Standard Deviation:", str(np.std(test_RMSE)))
print("Mean:",str(np.mean(test_R2)))
print("Standard Deviation:", str(np.std(test_R2)))
```