Rakshit_102083022_CO28_ASS1

- Github - https://github.com/rakshitgarg99/Rapids_data_science
- Kaggle - https://www.kaggle.com/rakshitgarg99/house-prediction/edit

Leaderboard

Raw Data Refresh

YOUR RECENT SUBMISSION

submission.csv
Submitted by Rakshit Garg · Submitted just now

Score: 0.31956

Jump to your leaderboard position

3873	**Rakshit Garg**	0.31956	19	1h

Your Best Entry!
Your most recent submission scored 0.31956, which is the same as your previous score. Keep trying!

In [232…
```python
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input direc

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you c
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

Load Libraries

In [233…
```python
import cupy as np
import cudf as pd
from matplotlib import pyplot as plt
```

In [234…
```python
# pd.set_option('display.max_rows',None)
# pd.set_option('display.max_columns',None)
```

In [235…
```python
train = pd.read_csv('/kaggle/input/house-prices-advanced-regression-techniques/train.csv')
test = pd.read_csv('/kaggle/input/house-prices-advanced-regression-techniques/test.csv')
```

In [236…
```python
print(f'train df shape is {train.shape}')
print(f'test df shape is {test.shape}')
```

Performing pre-processing and EDA on train and test dataset

In [237…
```python
train.head()
```

In [238…
```python
train.info()
```

In [239…
```python
missing_df = train.isnull().sum() / len(train)
missing_df[missing_df > 0.40]

missing_df_2 = test.isnull().sum() / len(test)
missing_df_2[missing_df_2 > 0.40]

columns = missing_df[missing_df > 0.40].index.to_pandas()
print(columns)
for c in columns:
    train.drop(columns=c,axis=1,inplace=True)

columns = missing_df_2[missing_df_2 > 0.40].index.to_pandas()
for c in columns:
    test.drop(columns=c,axis=1,inplace=True)
```

In [240…
```python
train.isnull().sum()
```

In [241…
```python
test.isnull().sum()
```

In [242…
```python
train.drop(columns='Id',axis=1,inplace=True)
```

In [243…
```python
numerical_features = [feature for feature in train.columns if train[feature].dtype != 'O']
print(f'Number of Numerical Features are {len(numerical_features)}')

categorical_features = [feature for feature in train.columns if train[feature].dtype == 'O']
print(f'Number of Categorical Features are {len(categorical_features)}')
```

In [244…
```python
year_features = [feature for feature in numerical_features if 'Year' in feature or 'Yr' in feature ]
year_features
```

In [245…
```python
for feature in year_features:
    print(feature,train[feature].unique())
```

In [246…
```python
## Numerical variables are usually of 2 type
### Continous variable and Discrete Variables

discrete_features = [feature for feature in numerical_features if len(train[feature].unique()) < 25 and feature
print("Discrete Variables Count: ",len(discrete_features))

continuous_features =  [feature for feature in numerical_features if feature not in discrete_features and featu
print("Continuous Variables Count: ",len(continuous_features))
```

In [247…
```python
# Plotting deiscrete features and SalePrice
for feature in discrete_features:
    data = train.copy()
    data.groupby(feature)['SalePrice'].median().to_pandas().plot.bar()
    plt.xlabel(feature)
    plt.ylabel('SalePrice')
    plt.show()
```

In [248…
```python
# Analysing the continuous values by creating histograms to understand the distribution

for feature in continuous_features:
    data = train.copy()
    data[feature].to_pandas().hist()
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.show()
```

Here, we observe that some of the features don't follow the gaussian distribution. We can apply log transformation further.

We will be using logarithmic transformation during feature scaling

In [249…
```python
train[categorical_features].head()
```

In [250…
```python
for feature in categorical_features:
    data = train.copy()
    print(f'The feature is {feature} and no of categories are {len(data[feature].unique())}')
```

In [251…
```python
# Realtionship between target variable and categorical features

for feature in categorical_features:
    train.groupby(feature)['SalePrice'].median().to_pandas().plot(kind='bar')
    plt.title(feature)
    plt.xlabel(feature)
    plt.ylabel('SalePrice')
    plt.show()
```

Handling Nan values

In [252…
```python
## Handling Categorical features which are missing

categorical_features_nan = [feature for feature in train.columns if train[feature].isnull().sum() > 0 and trair

## Replace missing value with a new label
def replace_missing_nan_cat(dataset,features):
    data = dataset.copy()
    data[features] = data[features].fillna('Missing')
    return data

train = replace_missing_nan_cat(train,categorical_features)
test = replace_missing_nan_cat(test,categorical_features)
```

In [253…
```python
print(train[categorical_features_nan].isnull().sum())
```

In [254…
```python
# Check for numerical variables the contains missing values
numerical_features_nan = [feature for feature in train.columns if train[feature].isnull().sum() > 0 and train[f
print(numerical_features_nan)

for feature in numerical_features_nan:
    train[feature] = train[feature].fillna(train[feature].median())
```

In [255…
```python
# Check for numerical variables in test data the contains missing values
numerical_features_nan = [feature for feature in test.columns if test[feature].isnull().sum() > 0 and test[feat
print(numerical_features_nan)

for feature in numerical_features_nan:
    test[feature] = test[feature].fillna(test[feature].median())
```

In [256…
```python
# Handling Temporal Variables (Date Time Variables)

for feature in ['YearBuilt','YearRemodAdd','GarageYrBlt']:
    train[feature] = train['YrSold'] - train[feature]
    test[feature] = test['YrSold'] - test[feature]

train[['YearBuilt','YearRemodAdd','GarageYrBlt']].head()
```

Handling Rare Categorical Feature We will remove categorical variables that are present less than 1% of the observations

In [257…
```python
for feature in categorical_features:
    temp = train.groupby(feature)['SalePrice'].count() / len(train)
    f_list = list(temp[temp>0.1].index.to_pandas())
    cnt=0
    for i in train[feature].to_pandas():
        if(i not in f_list):
            train[feature][cnt]='Others'
        cnt+=1


for feature in categorical_features:
    temp = test[feature].value_counts() / len(test)
    f_list = list(temp[temp>0.1].index.to_pandas())
    cnt=0
    for i in test[feature].to_pandas():
        if(i not in f_list):
            test[feature][cnt]='Others'
        cnt+=1
```

In [258…
```python
train['MSZoning'].unique()
test['MSZoning'].unique()
```

Feature Scaling

In [259…
```python
print(categorical_features)
```

In [260…
```python
# Encode the categorical variables
from cuml.preprocessing.LabelEncoder import LabelEncoder

enc = LabelEncoder()
for c in categorical_features:
    train[c] = enc.fit_transform(train[c])
    test[c] = enc.fit_transform(test[c])
```

Feature Selection We are selecting numerical features which have more than 0.50 or less than -0.50 correlation rate based on Pearson Correlation Method—which is the default value of parameter "method" in corr() function. As for selecting categorical features, I selected the categorical values which I believe have significant effect on the target variable such as Heating and MSZoning.

In [261…
```python
important_num_cols = list(train.corr()["SalePrice"][(train.corr()["SalePrice"]>0.50) | (train.corr()["SalePrice
cat_cols = ["MSZoning", "Utilities","BldgType","Heating","KitchenQual","SaleCondition","LandSlope"]
important_cols = important_num_cols + cat_cols

train = train[important_cols]
test_X = test[["OverallQual","YearBuilt","YearRemodAdd","ExterQual","TotalBsmtSF","1stFlrSF","GrLivArea","FullB
                 "MSZoning", "Utilities","BldgType","Heating","KitchenQual","SaleCondition","LandSlope"]]


len(train.columns)
```

Applying Regression

In [262…
```python
from cuml import LinearRegression
import cuml
```

In [263…
```python
from cuml.model_selection import train_test_split

X = train.drop('SalePrice',axis=1)
y = train['SalePrice']

X=X.astype('float64')
y=y.astype('float64')

# Splitting Training data to check validation and accuracy of model as per parameters
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.30, random_state=26)
```

In [264…
```python
df2 = pd.DataFrame()
mse = []
r2 = []
mae = []
```

In [265…
```python
algo = ["svd", "eig", "qr", "svd-qr", "svd-jacobi"]
for i in algo:
    lr = LinearRegression(fit_intercept = True, normalize = False, algorithm = i)
    reg = lr.fit(X_train,y_train)
    preds = lr.predict(X_val) # predictions

    mse.append(cuml.metrics.regression.mean_squared_error(y_val,preds))
    r2.append(cuml.metrics.regression.r2_score(y_val,preds))
    mae.append(cuml.metrics.regression.mean_absolute_error(y_val,preds))
```

In [266…
```python
df2['algo'] = algo
df2['mse'] = mse
df2['r2'] = r2
df2['mae'] = mae
df2
```

Predicting Results on Test Dataset

In [267…
```python
test_X=test_X.astype('float64')

# combining the training data again
new_x = pd.concat([X_train,X_val],axis=0)
new_y = pd.concat([y_train, y_val],axis=0)
```

In [268…
```python
lr = LinearRegression(fit_intercept = True, normalize = False, algorithm = "svd")
reg = lr.fit(new_x,new_y)
preds = lr.predict(test_X)
print(preds)
```

Submission

In [271…
```python
submission = pd.DataFrame()
submission['Id'] = range(1461,2920)
submission['SalePrice'] = preds
print(submission)
submission.to_csv('submission.csv', index=False)
```

In [ ]: