import pandas as pd

bike\_data = pd.read\_csv('data/daily-bike-share.csv')

bike\_data.head()

numeric\_features = ['temp', 'atemp', 'hum', 'windspeed']

bike\_data[numeric\_features + ['rentals']].describe()

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

label = bike\_data['rentals']

fig, ax = plt.subplots(2, 1, figsize = (9,12))

ax[0].hist(label, bins=100)

ax[0].set\_ylabel('Frequency')

ax[0].axvline(label.mean(), color='magenta', linestyle='dashed', linewidth=2)

ax[0].axvline(label.median(), color='cyan', linestyle='dashed', linewidth=2)

ax[1].boxplot(label, vert=False)

ax[1].set\_xlabel('Rentals')

fig.suptitle('Rental Distribution')

fig.show()

for col in numeric\_features:

fig = plt.figure(figsize=(9, 6))

ax = fig.gca()

feature = bike\_data[col]

feature.hist(bins=100, ax = ax)

ax.axvline(feature.mean(), color='magenta', linestyle='dashed', linewidth=2)

ax.axvline(feature.median(), color='cyan', linestyle='dashed', linewidth=2)

ax.set\_title(col)

plt.show()

from sklearn.metrics import mean\_squared\_error, r2\_score

mse = mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

rmse = np.sqrt(mse)

print("RMSE:", rmse)

r2 = r2\_score(y\_test, predictions)

print("R2:", r2)

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import make\_scorer, r2\_score

alg = GradientBoostingRegressor()

params = {

'learning\_rate': [0.1, 0.5, 1.0],

'n\_estimators' : [50, 100, 150]

}

score = make\_scorer(r2\_score)

gridsearch = GridSearchCV(alg, params, scoring=score, cv=3, return\_train\_score=True)

gridsearch.fit(X\_train, y\_train)

print("Best parameter combination:", gridsearch.best\_params\_, "\n")

model=gridsearch.best\_estimator\_

print(model, "\n")

predictions = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

rmse = np.sqrt(mse)

print("RMSE:", rmse)

r2 = r2\_score(y\_test, predictions)

print("R2:", r2)

plt.scatter(y\_test, predictions)

plt.xlabel('Actual Labels')

plt.ylabel('Predicted Labels')

plt.title('Daily Bike Share Predictions')

z = np.polyfit(y\_test, predictions, 1)

p = np.poly1d(z)

plt.plot(y\_test,p(y\_test), color='magenta')

plt.show()

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.linear\_model import LinearRegression

import numpy as np

numeric\_features = [6,7,8,9]

numeric\_transformer = Pipeline(steps=[

('scaler', StandardScaler())])

categorical\_features = [0,1,2,3,4,5]

categorical\_transformer = Pipeline(steps=[

('onehot', OneHotEncoder(handle\_unknown='ignore'))])

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numeric\_features),

('cat', categorical\_transformer, categorical\_features)])

pipeline = Pipeline(steps=[('preprocessor', preprocessor),

('regressor', GradientBoostingRegressor())])

model = pipeline.fit(X\_train, (y\_train))

print (model)

predictions = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

rmse = np.sqrt(mse)

print("RMSE:", rmse)

r2 = r2\_score(y\_test, predictions)

print("R2:", r2)

plt.scatter(y\_test, predictions)

plt.xlabel('Actual Labels')

plt.ylabel('Predicted Labels')

plt.title('Daily Bike Share Predictions')

z = np.polyfit(y\_test, predictions, 1)

p = np.poly1d(z)

plt.plot(y\_test,p(y\_test), color='magenta')

plt.show()

pipeline = Pipeline(steps=[('preprocessor', preprocessor),

('regressor', RandomForestRegressor())])

model = pipeline.fit(X\_train, (y\_train))

print (model, "\n")

predictions = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

rmse = np.sqrt(mse)

print("RMSE:", rmse)

r2 = r2\_score(y\_test, predictions)

print("R2:", r2)

plt.scatter(y\_test, predictions)

plt.xlabel('Actual Labels')

plt.ylabel('Predicted Labels')

plt.title('Daily Bike Share Predictions - Preprocessed')

z = np.polyfit(y\_test, predictions, 1)

p = np.poly1d(z)

plt.plot(y\_test,p(y\_test), color='magenta')

plt.show()