DataRead:

Father:CLASS Reader:

 # Function that reads data
def read(file_path):

Child:CLASS CsvReader:

 # Function that drops columns
def read_csv(file_path):
change:def read(file_path):

 # Function that Split
def split_df(df,trainSize=0.7):
    #split with test size
    train_data,text_data = train_test_split(df, trainSize)
    return train_data,test_data

DATA EXPLORATION:

Father:CLASS DataExplorer:

 # Function that Explore Data
def check(df, colum_name=[]):

Child:CLASS RangeDataExplorer:

#Function to check the range of values in each column
def check_range(df):

change:def check(df, colum_name=[]):

Child:CLASS NormalityDataExplorer:(alpha=0.05)

 # Function to check normality
def test_normality(df, column_name, alpha=0.05):

change:def check(df, colum_name=[]):

Child:CLASS MissingValuesDataExplorer:

 # Function that detects missing values per columns
def detect_missing_values(df):

change:def check(df, colum_name=[]):

Child:CLASS OutliersDataExplorer

 # Function to check outliers using quantiles
def check_outliers_quantiles(df):

change:def check(df, colum_name=[]):

DataProcess:

Father:CLASS DeleteProcessor:

 # Function that drops columns
def delete(df, columns_to_drop):

Child:CLASS DropDeleteProcessor:

 # Function that drops columns
def drop_columns(df, columns_to_drop):

change:def delete(df, columns_to_drop):

Father:CLASS FillProcessor:

 # Function that Fill columns
def fill(df, column_name):

Child:CLASS MeanFillProcessor:

 # Function to fill missing values with the mean
def column_fill_mean(df, column_name:str):

Change: def fill(df, column_name):

 # Handle outliers -> replace values with mean
def handle_outliers_mean(df, column_name:str):

Child:CLASS DistributionFillProcessor:

 # fill missing values by random from the distribution
def distributed_fill(df, column_name):

change:def fill(df, columns_name):

Child:CLASS KNNFillProcessor:

 # fill missing values with KNN
def column_fill_KNN(df, column_name:str):

def fill(df, columns_name):

 # Handle outliers -> replace values with KNN
def handle_outliers_KNN(df, column_name:str):

FEATURE:

CLASS StringConvertor:

 # Remove the string in the engine column
def convert_engine(column):

 # Remove the string in the power kms_driven
def convert_kms(column):

 # Remove the string in the power kms_driven
def convert_seats(seats_str):

 # Extract the first word
def keep_first_word(input_string):

CLASS OwnershipConvertor::

#data:(ownership):1st Owner,(car):a
#data:(ownership)2ndt Owner,(car):a
Result:
#data:(num_user):2,(car):a
#data:(num_user):2,(car):a

 # Remove the string in the ownership column
def extract_first_integer(ownership_str):

 # Create a new column 'num_users' by applying the extract_first_integer function
def process_ownership(df):

CLASS PriceUnitConvertor::

#Function that converts column with strings to numerical values only
def convert_comma_to_dot(column):
def convert_price(column):

Father:CLASS Encoder:

Child:CLASS BinaryEncoder:

#function for binary variables where we specify the true and false values
def columns_binary(df, column_names: list, true_value, false_value):

Child:CLASS CategoryOneHotEncoder:

#function for categorical variables
def one_hot_encode(df, columns_to_encode):

Father:CLASS Graphics:

 # Function to create Graphs for specified columns
def plot_graphs(df,columns):

Child:CLASS BoxPlotsGraphics:

 # Function to create boxplots for specified columns
def create_boxplots(df, columns):

change:def plot_graphs(df,columns):

Child:CLASS DistributionGraphics::

 # Plotting the distribution of the data
def plot_distribution(df, columns):

change:def plot_graphs(df,columns):

Child:CLASS CorrelationMatrixGraphics::

 # Correlation Matrix

def correlation_heatmap(df, columns):

change:def plot_graphs(df,columns):

Father:CLASS Transformator:

 # Function to transforms columns

def transform(df,column_name:str):

Child:CLASS Scaling_Normalization_Transformator:

 # Function to transforms columns

def scaling_normalization(df, column_name:str):

change:def transform(df,column_name:str):

Child:CLASS Scaling_Standardization_Transformator:

#Function to transforms columns

def scaling_standardization(df, column_name:str):

change:def transform(df,column_name:str):

Child:CLASS Log_Transformator:

 # Function to transforms columns

def log_transformation(df, column_name:str):

change:def transform(df,column_name:str):

Child:CLASS Square_Transformator:

 # Function to transforms columns

def square_transformation(df, column_name:str):

change:def transform(df,column_name:str):

MODEL:

Child:CLASS Linear_RegressionModel:

 # Linear Regression

def linear_regression(X_train, y_train, X_test, y_test):

change:def transform(X_train, y_train, X_test, y_test):

——: CLASS Model Comparator:

 # Function that runs every model
def best_model(X_train, y_train, X_test, y_test):

MODELCV:

Father:CLASS ModelCV:

 # Function to transforms columns
def regression_process(X_train, y_train, X_test, y_test):

Child:CLASS Lasso_Regression_cvModel:(cv=10)

#Lasso Regression with Cross Validation
def lasso_regression_cv(X_train, y_train, X_test, y_test):
change:def transform(X_train, y_train, X_test, y_test):

Child:CLASS ridge_regression_cvTransformator: (cv=10)

#Ridge Regression with Cross Validation
def ridge_regression_cv(X_train, y_train, X_test,y_test):
change:def transform(X_train, y_train, X_test, y_test):

Child:CLASS gradient_boosting_cvTransformator: (cv=10)

#Gradient Boosting Regression with Cross Validation
def gradient_boosting_cv(X_train, y_train, X_test, y_test):
change:def transform(X_train, y_train, X_test, y_test):