```python
import csv
import os

import numpy as np
import pandas as pd
from keras.callbacks import EarlyStopping
from keras.layers import Dense, Dropout
from sklearn.metrics import (
    explained_variance_score,
    mean_absolute_error,
    mean_absolute_percentage_error,
    mean_squared_error,
)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow import keras
from tqdm import tqdm, trange


my_path = rf"{os.getcwd()}\CIAN_diplom"

df_whole = pd.read_csv(rf"{my_path}\df.csv")


districts = sorted(list(set(df_whole["district"])))
activation_functions = {
    "relu": keras.layers.ReLU(),
    "leaky_relu": keras.layers.LeakyReLU(),
    "elu": keras.layers.ELU(),
}
optimizers = {
    "rmsprop": keras.optimizers.legacy.RMSprop(),
    "adam": keras.optimizers.legacy.Adam(),
    "amsgrad": keras.optimizers.legacy.Adam(amsgrad=True),
    "adamax": keras.optimizers.legacy.Adamax(),
}
nodes = [
    (400,),
    (500,),
    (600,),
    (200, 50),
    (200, 100),
    (400, 200),
    (400, 300),
    (500, 200),
    (500, 300),
    (600, 200),
    (600, 300),
    (200, 100, 50),
    (400, 200, 100),
    (400, 200, 200),
    (400, 300, 100),
    (400, 300, 200),
    (500, 300, 200),
    (500, 300, 100),
    (600, 300, 200),
    (600, 300, 100),
    (200, 200, 100, 50),
    (400, 200, 100, 50),
    (400, 300, 200, 50),
    (400, 300, 200, 100),
    (400, 300, 300, 50),
    (500, 300, 200, 50),
    (500, 300, 200, 100),
    (500, 400, 200, 50),
    (500, 400, 200, 100),
```

```
    (600, 300, 200, 50),
    (600, 300, 200, 100),
    (700, 500, 300, 200),
    (700, 500, 200, 100),
    (700, 600, 300, 200),
    (800, 600, 300, 200),
    (900, 600, 300, 200),
]

with open(
    rf"{my_path}\grid_search_districts.csv", "a", encoding="utf-8", newline=""
) as ouf:
    writer = csv.writer(ouf, delimiter="$", quotechar="|", quoting=csv.QUOTE_MINIMAL)
    writer.writerow(
        (
            "district",
            "nodes",
            "function",
            "optimizer",
            "cv_number",
            "MSE",
            "RMSE",
            "MAE",
            "MAPE",
            "var_score",
        )
    )


for distr in tqdm(districts, desc="District"):
    df = df_whole[df_whole["district"] == distr]
    df = pd.get_dummies(df, drop_first=True)

    X = df.drop("full_price", axis=1)
    y = df["full_price"]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42
    )

    scaler = MinMaxScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    for node in tqdm(nodes, desc="Nodes", leave=False):
        for f_name, func in tqdm(
            activation_functions.items(), desc="Functions", leave=False
        ):
            for opt_name, opt in tqdm(
                optimizers.items(), desc="Optimizers", leave=False
            ):
                for cv_counter in trange(3, desc="CV", leave=False):
                    if len(node) == 1:
                        model = keras.models.Sequential(
                            [
                                Dense(
                                    node[0],
                                    activation=func,
                                    kernel_initializer="he_uniform",
                                    input_dim=X_train.shape[1],
                                ),
                                Dense(1, activation="linear"),
                            ]
                        )

                    elif len(node) == 2:
                        model = keras.models.Sequential(
                            [
```

```
                                Dense(
                                    node[0],
                                    activation=func,
                                    kernel_initializer="he_uniform",
                                    input_dim=X_train.shape[1],
                                ),
                                Dense(
                                    node[1],
                                    activation=func,
                                    kernel_initializer="he_uniform",
                                ),
                                Dense(1, activation="linear"),
                            ]
                        )

                    elif len(node) == 3:
                        model = keras.models.Sequential(
                            [
                                Dense(
                                    node[0],
                                    activation=func,
                                    kernel_initializer="he_uniform",
                                    input_dim=X_train.shape[1],
                                ),
                                Dense(
                                    node[1],
                                    activation=func,
                                    kernel_initializer="he_uniform",
                                ),
                                Dropout(0.3, seed=123),
                                Dense(
                                    node[2],
                                    activation=func,
                                    kernel_initializer="he_uniform",
                                ),
                                Dense(1, activation="linear"),
                            ]
                        )

                    elif len(node) == 4:
                        model = keras.models.Sequential(
                            [
                                Dense(
                                    node[0],
                                    activation=func,
                                    kernel_initializer="he_uniform",
                                    input_dim=X_train.shape[1],
                                ),
                                Dense(
                                    node[1],
                                    activation=func,
                                    kernel_initializer="he_uniform",
                                ),
                                Dropout(0.3, seed=123),
                                Dense(
                                    node[2],
                                    activation=func,
                                    kernel_initializer="he_uniform",
                                ),
                                Dropout(0.3, seed=123),
                                Dense(
                                    node[3],
                                    activation=func,
                                    kernel_initializer="he_uniform",
                                ),
                                Dense(1, activation="linear"),
                            ]
```

```
                        )

                    model.compile(
                        loss="mse",
                        optimizer=opt,
                    )

                    early_stop = EarlyStopping(
                        monitor="val_loss", mode="min", verbose=0, patience=10
                    )

                    model.fit(
                        X_train,
                        y_train.values,
                        epochs=500,
                        batch_size=128,
                        validation_data=(X_test, y_test.values),
                        callbacks=[early_stop],
                        verbose=0,
                    )

                    predictions = model.predict(X_test, verbose=0)

                    del model

                    mae = mean_absolute_error(y_test, predictions)
                    mse = mean_squared_error(y_test, predictions)
                    rmse = np.sqrt(mean_squared_error(y_test, predictions))
                    mape = mean_absolute_percentage_error(y_test, predictions)
                    var_score = explained_variance_score(y_test, predictions)

                    data = (
                        distr,
                        node,
                        f_name,
                        opt_name,
                        cv_counter,
                        mse,
                        rmse,
                        mae,
                        mape,
                        var_score,
                    )

                    with open(
                        rf"{my_path}\grid_search_districts.csv",
                        "a",
                        encoding="utf-8",
                        newline="",
                    ) as ouf:
                        writer = csv.writer(
                            ouf,
                            delimiter="$",
                            quotechar="|",
                            quoting=csv.QUOTE_MINIMAL,
                        )
                        writer.writerow(data)
```