```python
from pathlib import Path
import os
import zipfile

from .consts import fuel_densities, owner_type_order

import pandas as pd
import kaggle


class DataLoader:
    """
    DataLoader that downloads dataset from kaggle competition `gist-mldl-25f-hw3` and proprocesses.

    Args:
      dir (Path | str): directory of the dataset. The directory should contain `train.csv` and `test.csv`.
    """

    def __init__(self, dir: Path | str):
        if isinstance(dir, str):
            dir = Path(dir)
        if not dir.exists():
            dir.mkdir(parents=True)
        self.dir = dir
        self.competition = "gist-mldl-25f-hw3"

    def load(self, download: bool = False) -> tuple[pd.DataFrame, pd.DataFrame]:
        """
        Loads training and test dataset and preprocesses.

        Args:
            download (bool): Whether to download the dataset from kaggle. Default to False.

        Returns:
          (DataFrame, DataFrame): A tuple of the loaded and preprocessed training and test dataset.
        """
        if download:
            self.download()

        df_train = pd.read_csv(self.dir / "train.csv", index_col="ID")
        df_test = pd.read_csv(self.dir / "test.csv", index_col="ID")

        df_train["_split"] = "train"
        df_test["_split"] = "test"

        df = pd.concat([df_train, df_test])

        df = self.clean(df)
        df = self.encode(df)

        df_train = df.loc[df["_split"] == "train"].drop(columns=["_split"])
        df_test = df.loc[df["_split"] == "test"].drop(columns=["_split"])

        df_train = self.impute(df_train)

        return df_train, df_test

    def download(self) -> tuple[Path, Path]:
        """
        Downloads dataset from kaggle competition.

        Args:
          competition (str): Kaggle competition to download dataset from.
          dir (Path): Path to download the dataset at.

        Returns:
          (Path, Path): Tuple of training dataset and test dataset paths.
        """
        kaggle.api.authenticate()
        kaggle.api.competition_download_files(self.competition, self.dir)
        zip_file_path = Path(self.dir, self.competition).with_suffix(".zip")
        with zipfile.ZipFile(zip_file_path, "r") as zip_ref:
            zip_ref.extractall(self.dir)
        os.remove(zip_file_path)

        return (self.dir / "train.csv", self.dir / "test.csv")

    def clean(self, df: pd.DataFrame) -> pd.DataFrame:
        """
        Cleans the data to fix any errors or inconsistencies.
        - Replaces `\\N` missing value indicators with `None`
        - Normalizes units (`Mileage`: km/kg to kmpl, `New_Price`: Cr to Lakh)
        - Extracts numeric values from text (`Engine`, `Power`)
        - Renames columns for consistency

        Args:
          df (DataFrame): The dataframe to be processed.

        Returns:
          DataFrame: The cleaned dataframe.
        """
        df = df.replace("\\N", None)

        df["Mileage"] = self._normalize_mileage(df)
        df["Engine"] = pd.to_numeric(
            df["Engine"].str.split(expand=True)[0], errors="coerce"
        )
        df["Power"] = pd.to_numeric(
            df["Power"].str.split(expand=True)[0], errors="coerce"
        )
        df["New_Price"] = self._normalize_new_price(df)

        df = df.rename(columns={"No. of Doors": "Doors"})

        return df

    def encode(self, df: pd.DataFrame) -> pd.DataFrame:
        """
        Encodes the statistical data types (numeric and categorical).
        - Sets appropriate data types for numeric columns
        - Sets categorical data types for categorical columns
        - Sets ordinal encoding for `Owner_Type`

        Args:
          df (DataFrame): The dataframe to be processed.

        Returns:
          DataFrame: The encoded dataframe.
        """
        df["Kilometers_Driven"] = pd.to_numeric(
            df["Kilometers_Driven"], errors="coerce"
        ).astype("Int64")
        df["Year"] = pd.to_numeric(df["Year"], errors="coerce").astype("Int64")
```

```
    df["Engine"] = df["Engine"].astype("Int64")
    df["Seats"] = pd.to_numeric(df["Seats"], errors="coerce").astype("Int64")
    df["Doors"] = pd.to_numeric(df["Doors"], errors="coerce").astype("Int64")

    df["Name"] = df["Name"].astype("category")
    df["Location"] = df["Location"].astype("category")
    df["Fuel_Type"] = df["Fuel_Type"].astype("category")
    df["Transmission"] = df["Transmission"].astype("category")
    df["Colour"] = df["Colour"].astype("category")

    ordinal_owner_category = pd.CategoricalDtype(
        categories=owner_type_order, ordered=True
    )
    df["Owner_Type"] = df["Owner_Type"].astype(ordinal_owner_category)

    return df

def impute(self, df: pd.DataFrame) -> pd.DataFrame:
    """
    Imputes any missing values.
    - Drops rows with missing features other than `Power` and `New_Price` (very few)
    - Other missing values kept as-is for further processing

    Args:
      df (DataFrame): The dataframe to be processed.

    Returns:
      DataFrame: The imputed dataframe.
    """
    return df.dropna(subset=list(set(df.columns) - {"Power", "New_Price"}))

def _normalize_mileage(self, df: pd.DataFrame) -> pd.Series:
    """
    Normalizes mileage by converting km/kg to kmpl based on fuel type density.

    Note: Even though this is not strictly in the realm of preprocessing, this is included here for simplicity. It is getting rid of inconsistency anyway.

    Args:
      df (DataFrame): The dataframe containing Mileage and Fuel_Type columns.
    Returns:
      Series: Normalized mileage values in kmpl.
    """
    mileage_split = df["Mileage"].str.split(expand=True)
    mileage_value = pd.to_numeric(mileage_split[0], errors="coerce")
    mileage_unit = mileage_split[1]

    conversion_factors = fuel_densities

    normalized_mileage = mileage_value.copy()

    for fuel_type, factor in conversion_factors.items():
        mask = (mileage_unit == "km/kg") & (df["Fuel_Type"] == fuel_type)
        normalized_mileage.loc[mask] = mileage_value.loc[mask] * factor

    return normalized_mileage

def _normalize_new_price(self, df: pd.DataFrame) -> pd.Series:
    """
    Normalizes New_Price to Lakh by converting Cr to Lakh (1 Cr = 100 Lakh).

    Args:
      df (DataFrame): The dataframe containing New_Price column.
    Returns:
      Series: Normalized price values in Lakh.
    """
    price_split = df["New_Price"].str.split(expand=True)
    price_value = pd.to_numeric(price_split[0], errors="coerce")
    price_unit = price_split[1]

    normalized_price = price_value.copy()
    cr_mask = price_unit == "Cr"
    normalized_price.loc[cr_mask] = price_value.loc[cr_mask] * 100

    return normalized_price
```