Module 4 exercises: Preprocessing Data for Analysis

Load in the dataset `renfe_trains.csv`

Initial data inspection

1. Inspect the columns of the DataFrame. Specifically, consider the type of each column and whether it seems reasonable. If not, investigate why.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 85948 entries, 0 to 85947
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   company        85948 non-null  object
 1   origin         85948 non-null  object
 2   destination    85948 non-null  object
 3   departure      85948 non-null  object
 4   arrival        85948 non-null  object
 5   vehicle_class  77116 non-null  object
 6   price          72769 non-null  object
 7   fare           77116 non-null  object
dtypes: object(8)
memory usage: 5.2+ MB
```

2. It seems like we have some bad values in the price column with the value ‘price’.

   You can see them by using the method .value_counts().

```
df['price'].value_counts()
```

```
price
price     3875
76.3      3794
85.1      3615
107.7     2845
53.4      2719
          ...
98.01        1
98.2         1
69.05        1
19.75        1
61.15        1
Name: count, Length: 389, dtype: int64
```

Inspect the specific rows where this is the case.

```
df[df['price']=='price']
```

	company	origin	destination	departure	arrival	vehicle_class	price	fare
69	company	origin	destination	departure	arrival	vehicle_class	price	fare
146	company	origin	destination	departure	arrival	vehicle_class	price	fare
209	company	origin	destination	departure	arrival	vehicle_class	price	fare
287	company	origin	destination	departure	arrival	vehicle_class	price	fare
347	company	origin	destination	departure	arrival	vehicle_class	price	fare
...	...	...	...	...	...	...	...	...

3. It looks like some sort of error has meant the column names have been fed into the data in intervals. Let's drop these rows as they are clearly an accident.

```
df = df[df['price'] != 'price']
```

4. We can now represent price using the appropriate type. Convert it to the appropriate data type.

```
df['price'] = df['price'].astype(np.float32)
```

Missing values

1. Identify whether there are missing values in the DataFrame.

```
df.isna().any()
```

```
company          False
origin           False
destination      False
departure        False
arrival          False
vehicle_class     True
price             True
fare              True
dtype: bool
```

```
df.isna().sum()
```

```
company              0
origin               0
destination          0
departure            0
arrival              0
vehicle_class     8832
price            13179
fare              8832
dtype: int64
```

2. Which columns are they in?

   Vehicle_class, price, fare

3. Inspect some rows which contain them.

```
df[df['price'].isna()]
```

	company	origin	destination	departure	arrival	vehicle_class	price	fare
11	renfe	MADRID	BARCELONA	2019-05-03 18:30:00	2019-05-03 21:20:00	Preferente	NaN	Promo
15	renfe	MADRID	BARCELONA	2019-04-23 07:30:00	2019-04-23 10:40:00	Turista	NaN	Promo
33	renfe	MADRID	SEVILLA	2019-04-21 21:25:00	2019-04-22 00:10:00	NaN	NaN	NaN
52	renfe	MADRID	SEVILLA	2019-04-17 09:45:00	2019-04-17 12:27:00	Turista	NaN	Flexible
65	renfe	MADRID	SEVILLA	2019-05-03 13:30:00	2019-05-03 16:05:00	Turista	NaN	Promo
...	...	...	...	...	...	...	...	...
85847	renfe	MADRID	SEVILLA	2020-11-22 09:00:00	2020-11-22 11:37:48	NaN	NaN	NaN
85850	renfe	MADRID	SEVILLA	2020-10-13 11:22:00	2020-10-13 16:05:12	NaN	NaN	NaN
85854	renfe	MADRID	BARCELONA	2020-11-06 10:30:00	2020-11-06 13:15:00	NaN	NaN	NaN
85866	renfe	MADRID	SEVILLA	2020-12-04 12:00:00	2020-12-04 14:31:48	NaN	NaN	NaN
85871	renfe	MADRID	SEVILLA	2020-10-13 11:22:00	2020-10-13 16:05:12	NaN	NaN	NaN

13179 rows × 8 columns

4. Drop all rows which have missing `vehicle_class` and `price` and `fare` (i.e. a value of NaN for all of them). Hint: how='all'

```
df.dropna(subset = ['vehicle_class', 'price', 'fare'],
          how='all',
          inplace=True
         )
```

5. Run the below code. What does it suggest about ticket price with respect to vehicle_class and fare?

   df[['vehicle_class', 'fare', 'price']].groupby(['vehicle_class', 'fare']).mean()

   There appears to be some influence of vehicle class and fare type on ticket price (as expected!)

6. Fill the remaining missing price values with the mean of all the prices.

   a. In the extension, you can try to tackle this more appropriately (and trickily!).

```
df.fillna({'price': df['price'].mean()},
          inplace=True)
```

7. Check you have gotten rid of all NaN values in df.

```
df.isna().sum()
```

```
company          0
origin           0
destination      0
departure        0
arrival          0
vehicle_class    0
price            0
fare             0
dtype: int64
```

Deduplication

1. Use `duplicated` to see whether the dataset contains any duplicated rows.

```
df[df.duplicated()]
```

	company	origin	destination	departure	arrival	vehicle_class	price	fare
39	renfe	MADRID	BARCELONA	2019-04-30 07:00:00	2019-04-30 09:30:00	Turista Plus	94.550003	Promo
71	renfe	MADRID	SEVILLA	2019-05-18 09:00:00	2019-05-18 11:38:00	Turista	76.300003	Flexible
83	renfe	MADRID	BARCELONA	2019-05-27 17:00:00	2019-05-27 19:30:00	Turista	88.949997	Promo
132	renfe	MADRID	BARCELONA	2019-05-10 08:30:00	2019-05-10 11:15:00	Turista	85.099998	Promo
174	renfe	MADRID	BARCELONA	2019-05-13 14:00:00	2019-05-13 16:30:00	Turista	68.650002	Promo
...	...	...	...	...	...	...	...	...

2. As the dataset constitutes ticket price search results, theres a good chance duplication has come about due to the data collection method. For example, there are many tickets available on each train.

   We would want to investigate this further, but to use the functionality, let's get rid of these duplicate rows.

```
df.drop_duplicates(inplace=True)
```

Outliers

Identify outliers in the price column. A common measure used to determine outliers is 1.5 * IQR above the upper quartile (Q3) or below the lower quartile (Q1)

```python
IQR = df['price'].quantile(0.75) - df['price'].quantile(0.25)

upper_bound =  df['price'].quantile(0.75) + (1.5 * IQR)
lower_bound =  df['price'].quantile(0.25) - (1.5 * IQR)

outliers = df[(df['price'] < lower_bound) | (df['price'] > upper_bound)]
```

Examine these outliers. Do they appear to be erroneous or is there a reason that they exist?

No apparent reason beyond expensive fare types (mesa, flexible etc.)

```python
(outliers['fare'].value_counts()/df['fare'].value_counts())
```

```
fare
Adulto ida                     NaN
Básica                         NaN
COD.PROMOCIONAL                NaN
Doble Familiar-Flexible        NaN
Flexible                  0.028281
Individual-Flexible       0.333333
Mesa                      0.800000
Promo                     0.003377
Promo +                        NaN
YOVOY                          NaN
Name: count, dtype: float64
```

```python
(outliers['vehicle_class'].value_counts()/df['vehicle_class'].value_counts()) * 100
```

```
vehicle_class
Cama G. Clase             20.000000
Cama Turista                    NaN
Preferente                 7.317768
PreferenteSólo plaza H          NaN
Turista                    0.004916
Turista Plus               0.694444
Turista PlusSólo plaza H        NaN
Turista con enlace              NaN
TuristaSólo plaza H             NaN
Name: count, dtype: float64
```

Training, testing, validation

Split the dataset into training and testing sets, assuming you are trying to predict `price`.

```python
from sklearn.model_selection import train_test_split

features = list(set(df.columns) - {'price'})
target   = ['price']

X_train, X_test, y_train, y_test = train_test_split(df[features],
                                                    df[target],
                                                    test_size=0.25,
                                                    random_state=42)
```

Scaling

Using `scikit-learn`'s `StandardScaler`, scale the price column.

```python
from sklearn.preprocessing import StandardScaler

target_scaler = StandardScaler()

y_train = target_scaler.fit_transform(y_train)
y_test = target_scaler.transform(y_test)
```

Encoding

Appropriately encode the destination column.

```python
from sklearn.preprocessing import OneHotEncoder

destination_encoder = OneHotEncoder(sparse_output=False)
X_train[destination_encoder.get_feature_names_out()] = destination_encoder.fit_transform(X_train[['destination']])
X_train.drop('destination', axis=1, inplace=True)
X_train
```

	arrival	company	departure	fare	vehicle_class	origin	destination_BARCELONA	destination_PONFERRADA	destination_SEVILLA
12815	2019-06-02 18:52:00	renfe	2019-06-02 14:40:00	Promo	Turista con enlace	MADRID	0.0	1.0	0.0
9927	2019-05-21 10:40:00	renfe	2019-05-21 07:30:00	Promo	Turista	MADRID	1.0	0.0	0.0
42431	2019-07-22 18:30:00	renfe	2019-07-22 16:00:00	Promo	Turista	MADRID	0.0	0.0	1.0
72039	2020-04-15 11:45:00	renfe	2020-04-15 09:00:00	Promo +	Turista	MADRID	1.0	0.0	0.0
8358	2019-04-21 20:40:00	renfe	2019-04-21 17:30:00	Flexible	Turista	MADRID	1.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...

```
X_test[destination_encoder.get_feature_names_out()] = destination_encoder.transform(X_test[['destination']])
X_test.drop('destination', axis=1, inplace=True)
X_test
```

	arrival	company	departure	fare	vehicle_class	origin	destination_BARCELONA	destination_PONFERRADA	destination_SEVILLA
55582	2019-12-10 18:33:00	renfe	2019-12-10 14:40:00	Promo	Turista con enlace	MADRID	0.0	1.0	0.0
25164	2019-07-11 17:30:00	renfe	2019-07-11 15:00:00	Promo	Turista	MADRID	1.0	0.0	0.0
32346	2019-06-06 23:55:00	renfe	2019-06-06 21:25:00	Promo	Turista	MADRID	1.0	0.0	0.0
22110	2019-07-02 21:30:00	renfe	2019-07-02 19:00:00	Promo	Turista	MADRID	1.0	0.0	0.0
56968	2020-02-24 08:40:00	renfe	2020-02-24 06:10:00	Promo	Preferente	MADRID	1.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...

Stretch exercises

As it appears `price` depends upon `vehicle_class` and `fare`, we choose to replace missing `price` values with the average for their `vehicle_class` and `fare` category. Write some code which does this.

```
df['price'].combine_first(df.groupby(by=['vehicle_class', 'fare'])['price'].transform(np.mean))
```

```
0         69.400002
1         43.549999
2         85.099998
3        107.699997
4        107.699997
            ...
85940     50.700001
85941     50.700001
```