preproc_final

April 28, 2023

[ ]:
```
# Imports
import pandas as pd
import swifter
import pickle
import matplotlib as plt
from datetime import datetime
```

[ ]:
```
# Creating csv file into dataframe
df = pd.read_csv("Data/price_paid_records.csv")
```

[ ]:
```
# Dropping irrelevant columns
df.drop('Transaction unique identifier', axis=1, inplace=True)
df.drop('Duration', axis=1, inplace=True)
df.drop('PPDCategory Type', axis=1, inplace=True)
df.drop('Record Status - monthly file only', axis=1, inplace=True)
df.drop('Old/New', axis=1, inplace=True)

# Dropping NULL Values
df.dropna(inplace=True)
```

[ ]:
```
# Removing time part of date of transfer
df["Date of Transfer"] = df["Date of Transfer"].swifter.apply(lambda x: x.
 ↪split(" ")[0])

# Splitting date of transfer into 3 columns for YYYY-MM-DD
df["year"] = df["Date of Transfer"].swifter.apply(lambda x: x.split("-")[0]).
 ↪astype("int")
df["month"] = df["Date of Transfer"].swifter.apply(lambda x: x.split("-")[1]).
 ↪astype("int")
df["day"] = df["Date of Transfer"].swifter.apply(lambda x: x.split("-")[2]).
 ↪astype("int")
```

```
Pandas Apply:   0%|          | 0/22489348 [00:00<?, ?it/s]
Pandas Apply:   0%|          | 0/22489348 [00:00<?, ?it/s]
Pandas Apply:   0%|          | 0/22489348 [00:00<?, ?it/s]
Pandas Apply:   0%|          | 0/22489348 [00:00<?, ?it/s]
```

[ ]:
```
# Renaiming feature to remove "/"
df.rename(columns={"Town/City":"City"}, inplace=True)
df.rename(columns={"Property Type":"property_type"}, inplace=True)
df.rename(columns={"Price":"price"}, inplace=True)
```

[ ]:
```
# Outcode classification pre-requisites
ocDict =   {
    'AB': 'Aberdeen',
    'AL': 'St Albans',
    'B': 'Birmingham',
    'BA': 'Bath',
    'BB': 'Blackburn',
    'BD': 'Bradford',
    'BH': 'Bournemouth',
    'BL': 'Bolton',
    'BN': 'Brighton',
    'BR': 'Bromley',
    'BS': 'Bristol',
    'BT': 'Belfast',
    'CA': 'Carlisle',
    'CB': 'Cambridge',
    'CF': 'Cardiff',
    'CH': 'Chester',
    'CM': 'Chelmsford',
    'CO': 'Colchester',
    'CR': 'Croydon',
    'CT': 'Canterbury',
    'CV': 'Coventry',
    'CW': 'Crewe',
    'DA': 'Dartford',
    'DD': 'Dundee',
    'DE': 'Derby',
    'DG': 'Dumfries',
    'DH': 'Durham',
    'DL': 'Darlington',
    'DN': 'Doncaster',
    'DT': 'Dorchester',
    'DY': 'Dudley',
    'E': 'London',
    'EC': 'London',
    'EH': 'Edinburgh',
    'EN': 'Enfield',
    'EX': 'Exeter',
    'FK': 'Falkirk',
    'FY': 'Blackpool',
    'G': 'Glasgow',
    'GL': 'Gloucester',
```

```
    'GU': 'Guildford',
    'HA': 'Harrow',
    'HD': 'Huddersfield',
    'HG': 'Harrogate',
    'HP': 'Hemel Hempstead',
    'HR': 'Hereford',
    'HS': 'Outer Hebrides',
    'HU': 'Hull',
    'HX': 'Halifax',
    'IG': 'Ilford',
    'IP': 'Ipswich',
    'IV': 'Inverness',
    'KA': 'Kilmarnock',
    'KT': 'Kingston upon Thames',
    'KW': 'Kirkwall',
    'KY': 'Kirkcaldy',
    'L': 'Liverpool',
    'LA': 'Lancaster',
    'LD': 'Llandrindod Wells',
    'LE': 'Leicester',
    'LL': 'Llandudno',
    'LN': 'Lincoln',
    'LS': 'Leeds',
    'LU': 'Luton',
    'M': 'Manchester',
    'ME': 'Rochester',
    'MK': 'Milton Keynes',
    'ML': 'Motherwell',
    'N': 'London',
    'NE': 'Newcastle upon Tyne',
    'NG': 'Nottingham',
    'NN': 'Northampton',
    'NP': 'Newport',
    'NR': 'Norwich',
    'NW': 'London',
    'OL': 'Oldham',
    'OX': 'Oxford',
    'PA': 'Paisley',
    'PE': 'Peterborough',
    'PH': 'Perth',
    'PL': 'Plymouth',
    'PO': 'Portsmouth',
    'PR': 'Preston',
    'RG': 'Reading',
    'RH': 'Redhill',
    'RM': 'Romford',
    'S': 'Sheffield',
```

```
    'SA': 'Swansea',
    'SE': 'London',
    'SG': 'Stevenage',
    'SK': 'Stockport',
    'SL': 'Slough',
    'SM': 'Sutton',
    'SN': 'Swindon',
    'SO': 'Southampton',
    'SP': 'Salisbury',
    'SR': 'Sunderland',
    'SS': 'Southend-on-Sea',
    'ST': 'Stoke-on-Trent',
    'SW': 'London',
    'SY': 'Shrewsbury',
    'TA': 'Taunton',
    'TD': 'Galashiels',
    'TF': 'Telford',
    'TN': 'Tunbridge Wells',
    'TQ': 'Torquay',
    'TR': 'Truro',
    'TS': 'Cleveland',
    'TW': 'Twickenham',
    'UB': 'Southall',
    'W': 'London',
    'WA': 'Warrington',
    'WC': 'London',
    'WD': 'Watford',
    'WF': 'Wakefield',
    'WN': 'Wigan',
    'WR': 'Worcester',
    'WS': 'Walsall',
    'WV': 'Wolverhampton',
    'YO': 'York',
    'ZE': 'Lerwick'
}

# Converting cities to all caps
for value in ocDict:
    ocDict[value] = ocDict[value].upper()

# Swapping keys and values
ocDict = {value: key for key, value in ocDict.items()}
```

[ ]:
```
# Making new dataframe for outcode values
oc_propdf = df.copy()
```

```
# Checking city value against postcode dictionary to check if it can be␣
 ↪categorised
oc_propdf["outcode"] = oc_propdf.swifter.apply(lambda x: ocDict[x.City] if x.
 ↪City in ocDict.keys() else None, axis=1)
```

```
Pandas Apply:   0%|          | 0/22489348 [00:00<?, ?it/s]
```

[ ]:
```
# Printing percentage of classified values and dropping nulls
print("Percentage unclassified under Out Code")
print(oc_propdf["outcode"].isna().sum() / len(oc_propdf) * 100)

# Dropping NULL values
oc_propdf.dropna(inplace=True)
```

```
Percentage unclassified under Out Code
54.96005931341362
```

[ ]:
```
# Out/Area Code classification pre-requisities
oacDict = pickle.load(open("Dumps/allPC.sav", "rb"))

# Function for dictionary checking
def getOutcode(x):
    # Check county for outcode city for area code
    if x.County in oacDict.keys():
        if x.City in oacDict[x.County].keys():
            pCode = oacDict[x.County][x.City]
            return pCode

    # Check district for outcode city for area code
    if x.District in oacDict.keys():
        if x.City in oacDict[x.District].keys():
            pCode = oacDict[x.District][x.City]
            return pCode

    # check city for outcode city for area code
    if x.City in oacDict.keys():
        if x.City in oacDict[x.City].keys():
            pCode = oacDict[x.City][x.City]
            return pCode

    return None
```

[ ]:
```
# Making new dataframe for out/area code values
oac_propdf = df.copy()

# Checking against dictionary function to classify records
oac_propdf["outcode"] = oac_propdf.swifter.apply(lambda x: getOutcode(x),␣
 ↪axis=1)
```

```
Pandas Apply:   0%|          | 0/22489348 [00:00<?, ?it/s]
```

[ ]:
```
# Printing percentage of classified values and dropping nulls
print("Percentage unclassified under Out/Area Code")
print(oac_propdf["outcode"].isna().sum() / len(oac_propdf) * 100)

# Dropping NULL values
oac_propdf.dropna(inplace=True)
```

```
Percentage unclassified under Out/Area Code
40.84106395614493
```

[ ]:
```
# Printing record lengths for both dataframes
print("Number of records for out code dataframe")
print(len(oc_propdf))
print()
print("Number of records for out/area code dataframe")
print(len(oac_propdf))
```

```
Number of records for out code dataframe
10129189

Number of records for out/area code dataframe
13304459
```

[ ]:
```
# Dropping remaining unneeded columns for oc_propdf
oc_propdf = oc_propdf.drop("Date of Transfer", axis=1)
oc_propdf = oc_propdf.drop("City", axis=1)
oc_propdf = oc_propdf.drop("District", axis=1)
oc_propdf = oc_propdf.drop("County", axis=1)
oc_propdf
```

[ ]:

	price	property_type	year	month	day	outcode
0	25000	T	1995	8	18	OL
4	18899	S	1995	6	23	WF
5	81750	S	1995	5	19	SP
16	34000	S	1995	7	31	S
18	55000	S	1995	3	31	SN
…	…	…	…	…	…	…
22489341	85000	T	2017	2	27	LS
22489343	175000	S	2017	2	20	LS
22489345	274000	D	2017	2	24	HD
22489346	36000	T	2017	2	22	HX
22489347	145000	T	2017	3	3	LS

[10129189 rows x 6 columns]

[ ]:
```
# Dropping remaining unneeded columns for oac_propdf
oac_propdf = oac_propdf.drop("Date of Transfer", axis=1)
oac_propdf = oac_propdf.drop("City", axis=1)
oac_propdf = oac_propdf.drop("District", axis=1)
oac_propdf = oac_propdf.drop("County", axis=1)
oac_propdf
```

[ ]:

	price	property_type	year	month	day	outcode
0	25000	T	1995	8	18	OL1
1	42500	S	1995	8	9	RM17
2	45000	T	1995	6	30	TA9
5	81750	S	1995	5	19	SP1
6	56000	S	1995	3	10	OX28
…	…	…	…	…	…	…
22489340	175000	D	2017	3	22	HX7
22489341	85000	T	2017	2	27	LS99
22489343	175000	S	2017	2	20	LS99
22489344	586945	D	2017	2	15	LS22
22489347	145000	T	2017	3	3	LS99

[13304459 rows x 6 columns]

[ ]:
```
# Adding time constraint values to both dataframe
oc_propdf["time_const"] = oc_propdf.swifter.apply(lambda x: x.year*365+x.
 ↪month*30+x.day, axis=1)
oac_propdf["time_const"] = oac_propdf.swifter.apply(lambda x: x.year*365+x.
 ↪month*30+x.day, axis=1)
```

[ ]:
```
# Encoding labelled values and saving encoders to dumps
from sklearn.preprocessing import LabelEncoder

# Type encoder
type_enc = LabelEncoder()

# Outcode encoder
oc_enc = LabelEncoder()
oc_propdf["outcode"] = oc_enc.fit_transform(oc_propdf["outcode"])
oc_propdf["property_type"] = type_enc.fit_transform(oc_propdf["property_type"])

# Out/Area encoder
oac_enc = LabelEncoder()
oac_propdf["outcode"] = oac_enc.fit_transform(oac_propdf["outcode"])
oac_propdf["property_type"] = type_enc.transform(oac_propdf["property_type"])

# Creating encoder dumps
pickle.dump(type_enc, open("Dumps/type_encoder.sav", "wb"))
pickle.dump(oc_enc, open("Dumps/oc_encoder.sav", "wb"))
```

```
pickle.dump(oac_enc, open("Dumps/oac_encoder.sav", "wb"))
```

[ ]:
```
# Printing final Out dataframe
oc_propdf
```

[ ]:

	price	property_type	year	month	day	outcode	time_const
0	25000	4	1995	8	18	58	728433
4	18899	3	1995	6	23	90	728378
5	81750	3	1995	5	19	75	728344
16	34000	3	1995	7	31	67	728416
18	55000	3	1995	3	31	73	728296
…	…	…	…	…	…	…	…
22489341	85000	4	2017	2	27	48	736292
22489343	175000	3	2017	2	20	48	736285
22489345	274000	0	2017	2	24	33	736289
22489346	36000	4	2017	2	22	38	736287
22489347	145000	4	2017	3	3	48	736298

[10129189 rows x 7 columns]

[ ]:
```
# Printing final Out/Area dataframe
oac_propdf
```

[ ]:

	price	property_type	year	month	day	outcode	time_const
0	25000	4	1995	8	18	562	728433
1	42500	3	1995	8	9	669	728424
2	45000	4	1995	6	30	807	728385
5	81750	3	1995	5	19	751	728344
6	56000	3	1995	3	10	577	728275
…	…	…	…	…	…	…	…
22489340	175000	0	2017	3	22	354	736317
22489341	85000	4	2017	2	27	479	736292
22489343	175000	3	2017	2	20	479	736285
22489344	586945	0	2017	2	15	475	736280
22489347	145000	4	2017	3	3	479	736298

[13304459 rows x 7 columns]

[ ]:
```
# Saving both dataframes back to dumps to be used in training
pickle.dump(oc_propdf, open("Dumps/oc_propdf.sav", "wb"))
pickle.dump(oac_propdf, open("Dumps/oac_propdf.sav", "wb"))
```