Can refer to this website

<https://www.kaggle.com/stefanoleone992/adm-project-dataset-cleaning>

import pandas as pd

import numpy as np

import seaborn as sns

pd.set\_option('display.max\_columns', None)

pd.options.mode.chained\_assignment = None

accidents\_short\_default = pd.read\_csv('../input/Accident\_Information.csv', low\_memory=False, index\_col=None, nrows=10)

accidents\_short\_default

accidents\_original\_headers = pd.read\_csv('../input/Accident\_Information.csv', low\_memory=False, index\_col=None, nrows=1)

list(accidents\_original\_headers)

accidents\_chunk = pd.read\_csv('../input/Accident\_Information.csv', low\_memory=False, index\_col=None, chunksize=200000)

chunk\_list = []

for chunk **in** accidents\_chunk:

chunk\_filter = chunk[

(chunk.Year.astype(int) >= 2010) &

(chunk.Year.astype(int) <= 2014) &

(chunk['Road\_Type'] != "Unknown") &

(chunk['Junction\_Detail'] != "Data missing or out of range") &

(chunk['Road\_Surface\_Conditions'] != "Data missing or out of range") &

(chunk['Weather\_Conditions'] != "Data missing or out of range") &

(chunk['Latitude'].notnull()) &

(chunk['Longitude'].notnull()) &

(chunk['Special\_Conditions\_at\_Site'] == "None")

]

chunk\_list.append(chunk\_filter)

accidents = pd.concat(chunk\_list)

sns.countplot(accidents['Number\_of\_Vehicles'])

accidents = accidents[accidents.Number\_of\_Vehicles.astype(int) <= 5]

def label\_serious\_or\_fatal\_accident(row):

if row['Accident\_Severity'] == 'Fatal':

return 'Fatal\_Serious'

elif row['Accident\_Severity'] == 'Serious':

return 'Fatal\_Serious'

else:

return row['Accident\_Severity']

accidents['Accident\_Seriousness'] = accidents.apply (lambda row: label\_serious\_or\_fatal\_accident(row), axis=1)

accidents.drop(['Accident\_Severity'], inplace=True, axis=1)

accidents.rename(columns={'Accident\_Seriousness': 'Accident\_Severity'}, inplace=True)

accidents['Datetime'] = pd.to\_datetime(accidents['Date'] + ' ' + accidents['Time'])

accidents.drop(['Date', 'Time'], inplace=True, axis=1)

accidents = accidents[(accidents['Datetime'].notnull())]

accidents.rename(columns={'Year': 'Year\_Number'}, inplace=True)

def label\_season(row):

if '2009-12-20' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2010-03-20':

return 4

elif '2010-03-20' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2010-06-21':

return 1

elif '2010-06-21' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2010-09-23':

return 2

elif '2010-09-23' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2010-12-21':

return 3

elif '2010-12-21' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2011-03-20':

return 4

elif '2011-03-20' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2011-06-21':

return 1

elif '2011-06-21' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2011-09-22':

return 2

elif '2011-09-22' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2011-12-22':

return 3

elif '2011-12-22' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2012-03-20':

return 4

elif '2012-03-20' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2012-06-21':

return 1

elif '2012-06-21' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2012-09-22':

return 2

elif '2012-09-22' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2012-12-22':

return 3

elif '2012-12-22' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2013-03-20':

return 4

elif '2013-03-20' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2013-06-20':

return 1

elif '2013-06-20' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2013-09-22':

return 2

elif '2013-09-22' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2013-12-21':

return 3

elif '2013-12-21' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2014-03-20':

return 4

elif '2014-03-20' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2014-06-21':

return 1

elif '2014-06-21' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2014-09-23':

return 2

elif '2014-09-23' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2014-12-21':

return 3

elif '2014-12-21' <= row['Datetime'].strftime('%Y-%m-**%d**') < '2015-03-20':

return 4

else:

return row['Datetime'].strftime('%Y-%m-**%d**')

accidents['Season'] = accidents.apply (lambda row: label\_season(row), axis=1)

def label\_month\_of\_year(row):

return row['Datetime'].strftime('%m').lstrip("0").replace(" 0", "")

accidents['Month\_of\_Year'] = accidents.apply (lambda row: label\_month\_of\_year(row), axis=1)

def label\_day\_of\_month(row):

return row['Datetime'].strftime('**%d**').lstrip("0").replace(" 0", "")

accidents['Day\_of\_Month'] = accidents.apply (lambda row: label\_day\_of\_month(row), axis=1)

accidents = accidents.rename(columns = {'Day\_of\_Week': 'Day'})

def label\_day\_of\_week(row):

if row['Day'] == 'Monday':

return 1

elif row['Day'] == 'Tuesday':

return 2

elif row['Day'] == 'Wednesday':

return 3

elif row['Day'] == 'Thursday':

return 4

elif row['Day'] == 'Friday':

return 5

elif row['Day'] == 'Saturday':

return 6

elif row['Day'] == 'Sunday':

return 7

else:

return row['Day']

accidents['Day\_of\_Week'] = accidents.apply (lambda row: label\_day\_of\_week(row), axis=1)

accidents.drop(['Day'], inplace=True, axis=1)

def label\_hour\_of\_day(row):

return round(int((int(row['Datetime'].strftime('%H'))\*60) + (int(row['Datetime'].strftime('%M'))))/1440, 3)

accidents['Hour\_of\_Day'] = accidents.apply (lambda row: label\_hour\_of\_day(row), axis=1)

def label\_junction(row):

if row['Junction\_Detail'] == 'Mini-roundabout':

return 'Roundabout'

else:

return row['Junction\_Detail']

accidents['Junction'] = accidents.apply (lambda row: label\_junction(row), axis=1)

accidents.drop(['Junction\_Detail'], inplace=True, axis=1)

accidents.rename(columns={'Junction': 'Junction\_Detail'}, inplace=True)

def label\_weather(row):

if row['Weather\_Conditions'] == 'Fine + high winds':

return 'Fine'

elif row['Weather\_Conditions'] == 'Fine no high winds':

return 'Fine'

elif row['Weather\_Conditions'] == 'Raining + high winds':

return 'Raining'

elif row['Weather\_Conditions'] == 'Raining no high winds':

return 'Raining'

elif row['Weather\_Conditions'] == 'Snowing + high winds':

return 'Snowing'

elif row['Weather\_Conditions'] == 'Snowing no high winds':

return 'Snowing'

else:

return row['Weather\_Conditions']

accidents['Weather'] = accidents.apply (lambda row: label\_weather(row), axis=1)

def label\_high\_wind(row):

if row['Weather\_Conditions'] == 'Fine + high winds':

return 'Yes'

elif row['Weather\_Conditions'] == 'Fine no high winds':

return 'No'

elif row['Weather\_Conditions'] == 'Raining + high winds':

return 'Yes'

elif row['Weather\_Conditions'] == 'Raining no high winds':

return 'No'

elif row['Weather\_Conditions'] == 'Snowing + high winds':

return 'Yes'

elif row['Weather\_Conditions'] == 'Snowing no high winds':

return 'No'

else:

return 'No'

accidents['High\_Wind'] = accidents.apply (lambda row: label\_high\_wind(row), axis=1)

accidents.drop(['Weather\_Conditions'], inplace=True, axis=1)

def label\_lights(row):

if row['Light\_Conditions'] == 'Darkness - lights lit':

return 'Darkness - lights'

elif row['Light\_Conditions'] == 'Darkness - lights unlit':

return 'Darkness - no lights'

elif row['Light\_Conditions'] == 'Darkness - no lighting':

return 'Darkness - no lights'

else:

return row['Light\_Conditions']

accidents['Lights'] = accidents.apply (lambda row: label\_lights(row), axis=1)

accidents.drop(['Light\_Conditions'], inplace=True, axis=1)

east\_midlands = [

'Amber Valley', 'Ashfield', 'Bassetlaw', 'Blaby', 'Bolsover', 'Boston', 'Broxtowe', 'Charnwood', 'Chesterfield',

'Corby', 'Daventry', 'Derby', 'Derbyshire Dales', 'East Lindsey', 'East Northamptonshire', 'Erewash', 'Gedling',

'Harborough', 'High Peak', 'Hinckley and Bosworth', 'Kettering', 'Leicester', 'Lincoln', 'Mansfield', 'Melton',

'Newark and Sherwood', 'North East Derbyshire', 'North Kesteven', 'North West Leicestershire', 'Northampton',

'Nottingham', 'Oadby and Wigston', 'Rushcliffe', 'Rutland', 'South Derbyshire', 'South Holland', 'South Kesteven',

'South Northamptonshire', 'Wellingborough', 'West Lindsey'

]

east\_england = [

'Babergh', 'Basildon', 'Bedford', 'Braintree', 'Breckland', 'Brentwood', 'Broadland', 'Broxbourne', 'Cambridge',

'Castle Point', 'Central Bedfordshire', 'Chelmsford', 'Colchester', 'Dacorum', 'East Cambridgeshire',

'East Hertfordshire', 'Epping Forest', 'Fenland', 'Forest Heath', 'Great Yarmouth', 'Harlow', 'Hertsmere',

'Huntingdonshire', 'Ipswich', "King's Lynn and West Norfolk", 'Luton', 'Maldon', 'Mid Suffolk', 'North Hertfordshire',

'North Norfolk', 'Norwich', 'Peterborough', 'Rochford', 'South Cambridgeshire', 'South Norfolk', 'Southend-on-Sea',

'St. Albans', 'St. Edmundsbury', 'Stevenage', 'Suffolk Coastal', 'Tendring', 'Three Rivers', 'Thurrock', 'Uttlesford',

'Watford', 'Waveney', 'Welwyn Hatfield'

]

london = [

'Barking and Dagenham', 'Barnet', 'Bexley', 'Brent', 'Bromley', 'Camden', 'City of London', 'Croydon', 'Ealing',

'Enfield', 'Greenwich', 'Hackney', 'Hammersmith and Fulham', 'Haringey', 'Harrow', 'Havering', 'Hillingdon', 'Hounslow',

'Islington', 'Kensington and Chelsea', 'Kingston upon Thames', 'Lambeth', 'Lewisham', 'London Airport (Heathrow)', 'Merton',

'Newham', 'Redbridge', 'Richmond upon Thames', 'Southwark', 'Sutton', 'Tower Hamlets', 'Waltham Forest', 'Wandsworth', 'Westminster'

]

north\_east\_england = [

'County Durham', 'Darlington', 'Gateshead', 'Hartlepool', 'Middlesbrough', 'Newcastle upon Tyne', 'North Tyneside',

'Northumberland', 'Redcar and Cleveland', 'South Tyneside', 'Stockton-on-Tees', 'Sunderland'

]

north\_west\_england = [

'Allerdale', 'Barrow-in-Furness', 'Blackburn with Darwen', 'Blackpool', 'Bolton', 'Burnley', 'Bury', 'Carlisle',

'Cheshire East', 'Cheshire West and Chester', 'Chorley', 'Copeland', 'Eden', 'Fylde', 'Halton', 'Hyndburn',

'Knowsley', 'Lancaster', 'Liverpool', 'Manchester', 'Oldham', 'Pendle', 'Preston', 'Ribble Valley', 'Rochdale',

'Rossendale', 'Salford', 'Sefton', 'South Lakeland', 'South Ribble', 'St. Helens', 'Stockport', 'Tameside',

'Trafford', 'Warrington', 'West Lancashire', 'Wigan', 'Wirral', 'Wyre'

]

scotland = [

'Aberdeen City', 'Aberdeenshire', 'Angus', 'Argyll and Bute', 'Clackmannanshire', 'Dumfries and Galloway',

'Dundee City', 'East Ayrshire', 'East Dunbartonshire', 'East Lothian', 'East Renfrewshire', 'Edinburgh, City of',

'Falkirk', 'Fife', 'Glasgow City', 'Highland', 'Inverclyde', 'Midlothian', 'Moray', 'North Ayrshire',

'North Lanarkshire', 'Orkney Islands', 'Perth and Kinross', 'Renfrewshire', 'Scottish Borders', 'Shetland Islands',

'South Ayrshire', 'South Lanarkshire', 'Stirling', 'West Dunbartonshire', 'West Lothian', 'Western Isles'

]

south\_east\_england = [

'Adur', 'Arun', 'Ashford', 'Aylesbury Vale', 'Basingstoke and Deane', 'Bracknell Forest', 'Brighton and Hove',

'Canterbury', 'Cherwell', 'Chichester', 'Chiltern', 'Crawley', 'Dartford', 'Dover', 'East Hampshire', 'Eastbourne',

'Eastleigh', 'Elmbridge', 'Epsom and Ewell', 'Fareham', 'Gosport', 'Gravesham', 'Guildford', 'Hart', 'Hastings',

'Havant', 'Horsham', 'Isle of Wight', 'Lewes', 'Maidstone', 'Medway', 'Mid Sussex', 'Milton Keynes', 'Mole Valley',

'New Forest', 'Oxford', 'Portsmouth', 'Reading', 'Reigate and Banstead', 'Rother', 'Runnymede', 'Rushmoor',

'Sevenoaks', 'Shepway', 'Slough', 'South Bucks', 'South Oxfordshire', 'Southampton', 'Spelthorne', 'Surrey Heath',

'Swale', 'Tandridge', 'Test Valley', 'Thanet', 'Tonbridge and Malling', 'Tunbridge Wells', 'Vale of White Horse',

'Waverley', 'Wealden', 'West Berkshire', 'West Oxfordshire', 'Winchester', 'Windsor and Maidenhead', 'Woking',

'Wokingham', 'Worthing', 'Wycombe'

]

south\_west\_england =[

'Bath and North East Somerset', 'Bournemouth', 'Bristol, City of', 'Cheltenham', 'Christchurch', 'Cornwall',

'Cotswold', 'East Devon', 'East Dorset', 'Exeter', 'Forest of Dean', 'Gloucester', 'Mendip', 'Mid Devon',

'North Devon', 'North Dorset', 'North Somerset', 'Plymouth', 'Poole', 'Purbeck', 'Sedgemoor', 'South Gloucestershire',

'South Hams', 'South Somerset', 'Stroud', 'Swindon', 'Taunton Deane', 'Teignbridge', 'Tewkesbury', 'Torbay',

'Torridge', 'West Devon', 'West Dorset', 'West Somerset', 'Weymouth and Portland', 'Wiltshire'

]

wales = [

'Blaenau Gwent', 'Bridgend', 'Caerphilly', 'Cardiff', 'Carmarthenshire', 'Ceredigion', 'Conwy', 'Denbighshire',

'Flintshire', 'Gwynedd', 'Isle of Anglesey', 'Merthyr Tydfil', 'Monmouthshire', 'Neath Port Talbot', 'Newport',

'Pembrokeshire', 'Powys', 'Rhondda, Cynon, Taff', 'Swansea', 'The Vale of Glamorgan', 'Torfaen', 'Wrexham'

]

west\_midlands = [

'Birmingham', 'Bromsgrove', 'Cannock Chase', 'Coventry', 'Dudley', 'East Staffordshire', 'Herefordshire, County of',

'Lichfield', 'Malvern Hills', 'Newcastle-under-Lyme', 'North Warwickshire', 'Nuneaton and Bedworth', 'Redditch',

'Rugby', 'Sandwell', 'Shropshire', 'Solihull', 'South Staffordshire', 'Stafford', 'Staffordshire Moorlands',

'Stoke-on-Trent', 'Stratford-upon-Avon', 'Tamworth', 'Telford and Wrekin', 'Walsall', 'Warwick', 'Wolverhampton',

'Worcester', 'Wychavon', 'Wyre Forest'

]

yorkshire\_and\_the\_humber = [

'Barnsley', 'Bradford', 'Calderdale', 'Craven', 'Doncaster', 'East Riding of Yorkshire', 'Hambleton', 'Harrogate',

'Kingston upon Hull, City of', 'Kirklees', 'Leeds', 'North East Lincolnshire', 'North Lincolnshire', 'Richmondshire',

'Rotherham', 'Ryedale', 'Scarborough', 'Selby', 'Sheffield', 'Wakefield', 'York'

]

def label\_region(row):

if row['Local\_Authority\_(District)'] **in** east\_midlands:

return 'East Midlands'

elif row['Local\_Authority\_(District)'] **in** east\_england:

return 'East England'

elif row['Local\_Authority\_(District)'] **in** london:

return 'London'

elif row['Local\_Authority\_(District)'] **in** north\_east\_england:

return 'North East England'

elif row['Local\_Authority\_(District)'] **in** north\_west\_england:

return 'North West England'

elif row['Local\_Authority\_(District)'] **in** scotland:

return 'Scotland'

elif row['Local\_Authority\_(District)'] **in** south\_east\_england:

return 'South East England'

elif row['Local\_Authority\_(District)'] **in** south\_west\_england:

return 'South West England'

elif row['Local\_Authority\_(District)'] **in** wales:

return 'Wales'

elif row['Local\_Authority\_(District)'] **in** west\_midlands:

return 'Wast Midlands'

elif row['Local\_Authority\_(District)'] **in** yorkshire\_and\_the\_humber:

return 'Yorkshire and the Humber'

accidents['Region'] = accidents.apply (lambda row: label\_region(row), axis=1)

accidents.drop(['Local\_Authority\_(District)'], inplace=True, axis=1)

vehicles\_short\_default = pd.read\_csv('../input/Vehicle\_Information.csv', low\_memory=False, index\_col=None, nrows=10)

vehicles\_short\_default

vehicles\_original\_headers = pd.read\_csv('../input/Vehicle\_Information.csv', low\_memory=False, index\_col=None, nrows=1)

list(vehicles\_original\_headers)

vehicles\_chunk = pd.read\_csv('../input/Vehicle\_Information.csv', low\_memory=False, index\_col=None, chunksize=200000,

encoding = "ISO-8859-1")

chunk\_list = []

for chunk **in** vehicles\_chunk:

chunk\_list.append(chunk)

vehicles = pd.concat(chunk\_list)

accidents\_interesting\_list = accidents['Accident\_Index'].tolist()

vehicles = vehicles[vehicles['Accident\_Index'].isin(accidents\_interesting\_list)]

vehicles\_uninteresting = vehicles[

(vehicles['Age\_Band\_of\_Driver'] == 'Data missing or out of range') |

(vehicles['Age\_Band\_of\_Driver'] == '0 - 5') |

(vehicles['Age\_Band\_of\_Driver'] == '6 - 10') |

(vehicles['Age\_Band\_of\_Driver'] == '11 - 15') |

(vehicles['Age\_of\_Vehicle'].isnull()) |

(vehicles['Driver\_IMD\_Decile'].isnull()) |

(vehicles['Engine\_Capacity\_.CC.'].isnull()) |

(vehicles['Propulsion\_Code'] == 'Electric diesel') |

(vehicles['Propulsion\_Code'] == 'Fuel cells') |

(vehicles['Propulsion\_Code'] == 'Gas') |

(vehicles['Propulsion\_Code'] == 'Gas diesel') |

(vehicles['Propulsion\_Code'] == 'Gas/Bi-fuel') |

(vehicles['Propulsion\_Code'] == 'Hybrid electric') |

(vehicles['Propulsion\_Code'] == 'New fuel technology') |

(vehicles['Propulsion\_Code'] == 'Petrol/Gas (LPG)') |

(vehicles['make'].isnull()) |

(vehicles['Driver\_Home\_Area\_Type'] == 'Data missing or out of range') |

(vehicles['Sex\_of\_Driver'] == 'Not known') |

(vehicles['Sex\_of\_Driver'] == 'Data missing or out of range') |

(vehicles['X1st\_Point\_of\_Impact'] == 'Data missing or out of range') |

(vehicles['Vehicle\_Manoeuvre'] == 'Data missing or out of range') |

(vehicles['Junction\_Location'] == 'Data missing or out of range')

]

accidents\_uninteresting\_list = vehicles\_uninteresting['Accident\_Index'].tolist()

accidents = accidents[~accidents['Accident\_Index'].isin(accidents\_uninteresting\_list)]

sns.countplot(vehicles['Age\_of\_Vehicle'])

sns.countplot(vehicles['Engine\_Capacity\_.CC.'])

accidents\_interesting\_list = accidents['Accident\_Index'].tolist()

vehicles = vehicles[vehicles['Accident\_Index'].isin(accidents\_interesting\_list)]

vehicles\_uninteresting = vehicles[

(vehicles.Age\_of\_Vehicle.astype(int) > 30) |

(vehicles.Driver\_IMD\_Decile.astype(int) > 10) |

(vehicles['Engine\_Capacity\_.CC.'].astype(int) > 3500)

]

accidents\_uninteresting\_list = vehicles\_uninteresting['Accident\_Index'].tolist()

accidents = accidents[~accidents['Accident\_Index'].isin(accidents\_uninteresting\_list)]

def label\_driver\_journey\_purpose(row):

if row['Journey\_Purpose\_of\_Driver'] == 'Other/Not known (2005-10)':

return 'Other/Not known'

elif row['Journey\_Purpose\_of\_Driver'] == 'Not known':

return 'Other/Not known'

elif row['Journey\_Purpose\_of\_Driver'] == 'Other':

return 'Other/Not known'

elif row['Journey\_Purpose\_of\_Driver'] == 'Data missing or out of range':

return 'Other/Not known'

else:

return row['Journey\_Purpose\_of\_Driver']

vehicles['Driver\_Journey\_Purpose'] = vehicles.apply (lambda row: label\_driver\_journey\_purpose(row), axis=1)

vehicles.drop(['Journey\_Purpose\_of\_Driver'], inplace=True, axis=1)

def label\_age\_of\_driver(row):

if row['Age\_Band\_of\_Driver'] == '16 - 20':

return 1

elif row['Age\_Band\_of\_Driver'] == '21 - 25':

return 2

elif row['Age\_Band\_of\_Driver'] == '26 - 35':

return 3

elif row['Age\_Band\_of\_Driver'] == '36 - 45':

return 4

elif row['Age\_Band\_of\_Driver'] == '46 - 55':

return 5

elif row['Age\_Band\_of\_Driver'] == '56 - 65':

return 6

elif row['Age\_Band\_of\_Driver'] == '66 - 75':

return 7

elif row['Age\_Band\_of\_Driver'] == 'Over 75':

return 8

vehicles['Age\_of\_Driver'] = vehicles.apply (lambda row: label\_age\_of\_driver(row), axis=1)

vehicles.drop(['Age\_Band\_of\_Driver'], inplace=True, axis=1)

vehicles.rename(columns={'Engine\_Capacity\_.CC.': 'Engine\_CC'}, inplace=True)

def label\_make(row):

if row['make'] == 'FORD':

return 'Ford'

elif row['make'] == 'VAUXHALL':

return 'Vauxhall'

elif row['make'] == 'PEUGEOT':

return 'Peugeot'

elif row['make'] == 'VOLKSWAGEN':

return 'Volkswagen'

elif row['make'] == 'RENAULT':

return 'Renault'

elif row['make'] == 'HONDA':

return 'Honda'

elif row['make'] == 'TOYOTA':

return 'Toyota'

elif row['make'] == 'MERCEDES':

return 'Mercedes'

elif row['make'] == 'CITROEN':

return 'Citroen'

elif row['make'] == 'BMW':

return 'BMW'

elif row['make'] == 'NISSAN':

return 'Nissan'

elif row['make'] == 'AUDI':

return 'Audi'

elif row['make'] == 'FIAT':

return 'Fiat'

elif row['make'] == 'VOLVO':

return 'Volvo'

elif row['make'] == 'SUZUKI':

return 'Suzuki'

elif row['make'] == 'SKODA':

return 'Skoda'

elif row['make'] == 'YAMAHA':

return 'Yamaha'

elif row['make'] == 'LAND ROVER':

return 'Land Rover'

elif row['make'] == 'SEAT':

return 'Seat'

elif row['make'] == 'HYUNDAI':

return 'Hyundai'

elif row['make'] == 'MAZDA':

return 'Mazda'

elif row['make'] == 'ROVER':

return 'Rover'

elif row['make'] == 'MINI':

return 'Mini'

elif row['make'] == 'KIA':

return 'Kia'

elif row['make'] == '':

return ''

else:

return 'Other'

vehicles['Vehicle\_Make'] = vehicles.apply (lambda row: label\_make(row), axis=1)

vehicles.drop(['make'], inplace=True, axis=1)

def label\_vehicle\_category(row):

if row['Vehicle\_Type'] == 'Bus or coach (17 or more pass seats)':

return 'Bus/minibus'

elif row['Vehicle\_Type'] == 'Minibus (8 - 16 passenger seats)':

return 'Bus/minibus'

elif row['Vehicle\_Type'] == 'Taxi/Private hire car':

return 'Taxi'

elif row['Vehicle\_Type'] == 'Van / Goods 3.5 tonnes mgw or under':

return 'Van'

elif row['Vehicle\_Type'] == 'Motorcycle 125cc and under':

return 'Motorcycle'

elif row['Vehicle\_Type'] == 'Motorcycle 50cc and under':

return 'Motorcycle'

elif row['Vehicle\_Type'] == 'Motorcycle over 125cc and up to 500cc':

return 'Motorcycle'

elif row['Vehicle\_Type'] == 'Motorcycle over 500cc':

return 'Motorcycle'

elif row['Vehicle\_Type'] == 'Motorcycle - unknown cc':

return 'Motorcycle'

elif row['Vehicle\_Type'] == 'Agricultural vehicle':

return 'Other'

elif row['Vehicle\_Type'] == 'Electric motorcycle':

return 'Other'

elif row['Vehicle\_Type'] == 'Goods 7.5 tonnes mgw and over':

return 'Other'

elif row['Vehicle\_Type'] == 'Goods over 3.5t. and under 7.5t':

return 'Other'

elif row['Vehicle\_Type'] == 'Goods vehicle - unknown weight':

return 'Other'

elif row['Vehicle\_Type'] == 'Other vehicle':

return 'Other'

elif row['Vehicle\_Type'] == 'Pedal cycle':

return 'Other'

elif row['Vehicle\_Type'] == 'Ridden horse':

return 'Other'

elif row['Vehicle\_Type'] == 'Tram':

return 'Other'

elif row['Vehicle\_Type'] == 'Data missing or out of range':

return 'Other'

elif row['Vehicle\_Type'] == 'Mobility scooter':

return 'Other'

else:

return row['Vehicle\_Type']

vehicles['Vehicle\_Category'] = vehicles.apply (lambda row: label\_vehicle\_category(row), axis=1)

vehicles.drop(['Vehicle\_Type'], inplace=True, axis=1)

def label\_vehicle\_maneuver(row):

if row['Vehicle\_Manoeuvre'] == 'Changing lane to left':

return 'Changing lane'

elif row['Vehicle\_Manoeuvre'] == 'Changing lane to right':

return 'Changing lane'

elif row['Vehicle\_Manoeuvre'] == 'Going ahead left-hand bend':

return 'Going ahead'

elif row['Vehicle\_Manoeuvre'] == 'Going ahead other':

return 'Going ahead'

elif row['Vehicle\_Manoeuvre'] == 'Going ahead right-hand bend':

return 'Going ahead'

elif row['Vehicle\_Manoeuvre'] == 'Overtaking - nearside':

return 'Overtaking'

elif row['Vehicle\_Manoeuvre'] == 'Overtaking moving vehicle - offside':

return 'Overtaking'

elif row['Vehicle\_Manoeuvre'] == 'Overtaking static vehicle - offside':

return 'Overtaking'

elif row['Vehicle\_Manoeuvre'] == 'Waiting to go - held up':

return 'Waiting to go'

elif row['Vehicle\_Manoeuvre'] == 'Waiting to turn left':

return 'Waiting to go'

elif row['Vehicle\_Manoeuvre'] == 'Waiting to turn right':

return 'Waiting to go'

else:

return row['Vehicle\_Manoeuvre']

vehicles['Vehicle\_Maneuver'] = vehicles.apply (lambda row: label\_vehicle\_maneuver(row), axis=1)

vehicles.drop(['Vehicle\_Manoeuvre'], inplace=True, axis=1)

vehicles.rename(columns={'Vehicle\_Maneuver': 'Vehicle\_Manoeuvre'}, inplace=True

accidents = accidents.set\_index('Accident\_Index')

vehicles = vehicles.set\_index('Accident\_Index')

accidents\_categorical = pd.merge(accidents, vehicles, on='Accident\_Index')

accidents\_categorical.head(10)

columns\_to\_drop = ['1st\_Road\_Number', '2nd\_Road\_Class', '2nd\_Road\_Number', 'Carriageway\_Hazards',

'Did\_Police\_Officer\_Attend\_Scene\_of\_Accident', 'Junction\_Control',

'Local\_Authority\_(Highway)', 'Location\_Easting\_OSGR', 'Location\_Northing\_OSGR',

'LSOA\_of\_Accident\_Location', 'Pedestrian\_Crossing-Human\_Control',

'Pedestrian\_Crossing-Physical\_Facilities', 'Police\_Force', 'Number\_of\_Casualties',

'Special\_Conditions\_at\_Site', 'InScotland', 'Hit\_Object\_in\_Carriageway',

'Hit\_Object\_off\_Carriageway', 'model', 'Skidding\_and\_Overturning',

'Towing\_and\_Articulation', 'Vehicle\_Leaving\_Carriageway', 'Vehicle\_Location.Restricted\_Lane',

'Vehicle\_Reference', 'Was\_Vehicle\_Left\_Hand\_Drive', 'Year']

accidents\_categorical = accidents\_categorical.drop(columns\_to\_drop, axis='columns')

accidents\_categorical['Speed\_limit'] = accidents\_categorical['Speed\_limit'].astype(int)

accidents\_categorical['Age\_of\_Vehicle'] = accidents\_categorical['Age\_of\_Vehicle'].astype(int)

accidents\_categorical['Engine\_CC'] = accidents\_categorical['Engine\_CC'].astype(int)

accidents\_categorical['Driver\_IMD\_Decile'] = accidents\_categorical['Driver\_IMD\_Decile'].astype(int)

accidents\_categorical.rename(columns={'Year\_Number': 'Year'}, inplace=True)

accidents\_categorical['Accident\_Index'] = accidents\_categorical.index

cols = [

'Accident\_Index',

'Latitude',

'Longitude',

'Region',

'Urban\_or\_Rural\_Area',

'1st\_Road\_Class',

'Driver\_IMD\_Decile',

'Speed\_limit',

'Road\_Type',

'Road\_Surface\_Conditions',

'Weather',

'High\_Wind',

'Lights',

'Datetime',

'Year',

'Season',

'Month\_of\_Year',

'Day\_of\_Month',

'Day\_of\_Week',

'Hour\_of\_Day',

'Number\_of\_Vehicles',

'Age\_of\_Driver',

'Age\_of\_Vehicle',

'Junction\_Detail',

'Junction\_Location',

'X1st\_Point\_of\_Impact',

'Driver\_Journey\_Purpose',

'Engine\_CC',

'Propulsion\_Code',

'Vehicle\_Make',

'Vehicle\_Category',

'Vehicle\_Manoeuvre',

'Accident\_Severity'

]

accidents\_categorical = accidents\_categorical[cols]

accidents\_categorical.head(10)