Load and Combine Datasets

```
import numpy as np
import pandas as pd
```

The Census Profile contains over 2600 "characteristics" for each region, stored in a one-row-per-characteristic format, with columns representing various metadata and breakdowns by gender. In this section, we will select only those characteristics that correspond to our desired feature domains (age, housing, income, family & citizenship, education, employment, and commuting patterns). These are then normalized over the population/dwelling count, and transposed into the expected columnar format for further analysis as features. Characteristics in the Census Profile often have a hierarchical structure, and the provided names may not make sense outside of this. We will rename these variables in order to properly reflect their meaning.

```
# Assigning meaningful variable names to Census Profile characteristics in place of numeric IDs
all_vars = ["total_pop","total_dwellings","num_male",
            "age_25","age_30","age_35","age_40","age_45","age_50","age_55","age_60","age_65_up","age_mean","age_med",
            "homes_detached_house","homes_semidetached_house","homes_rowhouse","homes_duplex_apt","homes_lowrise_apt",
            "homes_highrise_apt","homes_other_stationary","homes_mobile",
            "avg_ppl_household","married_ppl","single_ppl",
            "avg_total_income","avg_aftertax_income","med_fulltime_income","avg_fulltime_income",
            "income_none","income_under_10k","income_10k","income_20k","income_30k","income_40k","income_50k","income_6(
            "income_70k","income_80k","income_90k","income_100k_up",
            "med_total_household_income","med_aftertax_household_income",
            "indigenous_ppl","nonindigenous_ppl",
            "home_owner","home_renter","home_gov_or_ind_band",
            "condo","non_condo","avg_rooms_home",
            "home_cost_under_30pct","home_cost_over_30pct",
            "can_citizen_ppl","non_citizen_ppl",
            "non_immigrant_ppl","immigrant_ppl","non_perm_res_ppl",
            "no_move_last_yr","moved_last_yr","no_move_5yrs","moved_last_5yrs",
            "school_no_hs","school_hs","school_college","school_uni_degree",
            "edu_field_education","edu_field_arts_comms","edu_field_humanities","edu_field_socsci_law","edu_field_bus_ad
            "edu_field_science","edu_field_math_cs","edu_field_engin_arch","edu_field_agri_res_env","edu_field_health",
            "edu_field_pers_protect_transp","edu_field_other",
            "workforce_participation_rate","workforce_employment_rate","workforce_unemployment_rate",
            "work_lastyear_didnotwork","work_lastyear_worked","work_lastyear_fulltime","work_lastyear_parttime","work_la
            "worktype_employee","worktype_selfemp",
            "occup_cat_snr_mgmt","occup_cat_busfin","occup_cat_science","occup_cat_health","occup_cat_edu_law_socserv",'
            "occup_cat_sales_serv","occup_cat_trades_transp","occup_cat_natres_agr","occup_cat_manuf_util",
            "occup_ind_agr_forest","occup_ind_mine_og","occup_ind_util","occup_ind_constr","occup_ind_manuf","occup_ind_
            "occup_ind_retail_trd","occup_ind_transp_warehs","occup_ind_info_culture","occup_ind_fin_insure","occup_ind_
            "occup_ind_prof_sci_tech_serv","occup_ind_mgmt","occup_ind_admsupport_wastemgmt","occup_ind_edu","occup_ind_
            "occup_ind_arts_ent_rec","occup_ind_accom_food_svc","occup_ind_other","occup_ind_pubadmin",
            "work_loc_home","work_loc_foreign","work_loc_notfixed","work_loc_workplace",
            "commute_same_subdiv","commute_same_div","commute_same_prov","commute_diff_prov",
            "commute_transp_cardriver","commute_transp_carpass","commute_transp_pubtrans","commute_transp_walk",
            "commute_transp_bike","commute_transp_other",
            "commute_time_under15","commute_time_15","commute_time_30","commute_time_45","commute_time_over60",
            "commute_start_5am","commute_start_6am","commute_start_7am","commute_start_8am","commute_start_9am","commute

var_names = dict(zip(
        [1,4,8,16,17,18,19,20,21,22,23,24,39,40,
          42,43,44,45,46,47,48,49,57,59,66,128,130,143,144,
          156,158,159,160,161,162,163,164,165,166,167,168,
          243,244,1403,1410,1415,1416,1417,1419,1420,1433,
          1466,1467,1523,1526,1528,1529,1537,1975,1976,1984,1985,
          2015,2016,2018,2024,2095,2097,2100,2109,2117,2121,2127,2132,2140,2143,
          2149,2155,2228,2229,2230,2232,2233,2234,2235,2236,2240,2245,
          2249,2250,2251,2252,2253,2254,2255,2256,2257,2258,
          2262,2263,2264,2265,2266,2267,2268,2269,2270,
          2271,2272,2273,2274,2275,2276,2277,2278,2279,2280,2281,
          2594,2595,2596,2597,
          2599,2600,2601,2602,
          2605,2606,2607,2608,2609,2610,
          2612,2613,2614,2615,2616,
          2618,2619,2620,2621,2622,2623],
        all_vars
))

# Divide variables into those that represent counts of people and dwellings, in order to divide by tl
ppl_vars = ["num_male","age_25","age_30","age_35","age_40","age_45","age_50","age_55","age_60","age_65_up",
                          "married_ppl","single_ppl","income_none","income_under_10k","income_10k","income_20k","income_
                          "income_40k","income_50k","income_60k","income_70k","income_80k","income_90k","income_100k_up'
                          "indigenous_ppl","nonindigenous_ppl","home_owner","home_renter","home_gov_or_ind_band",
                          "home_cost_under_30pct","home_cost_over_30pct","can_citizen_ppl","non_citizen_ppl","non_immig1
                          "immigrant_ppl","non_perm_res_ppl","no_move_last_yr","moved_last_yr","no_move_5yrs","moved_las
                          "school_no_hs","school_hs","school_college","school_uni_degree","edu_field_education","edu_fie
                          "edu_field_humanities","edu_field_socsci_law","edu_field_bus_admin","edu_field_science","edu_:
```

```
                         "edu_field_engin_arch","edu_field_agri_res_env","edu_field_health","edu_field_pers_protect_tra
                         "work_lastyear_didnotwork","work_lastyear_worked","work_lastyear_fulltime","work_lastyear_par
                         "worktype_employee","worktype_selfemp","occup_cat_snr_mgmt","occup_cat_busfin","occup_cat_sci
                         "occup_cat_edu_law_socserv","occup_cat_arts_rec","occup_cat_sales_serv","occup_cat_trades_tra
                         "occup_cat_manuf_util","occup_ind_agr_forest","occup_ind_mine_og","occup_ind_util","occup_ind_
                         "occup_ind_wholesale_trd","occup_ind_retail_trd","occup_ind_transp_warehs","occup_ind_info_cu
                         "occup_ind_realestate","occup_ind_prof_sci_tech_serv","occup_ind_mgmt","occup_ind_admsupport_
                         "occup_ind_health_socasst","occup_ind_arts_ent_rec","occup_ind_accom_food_svc","occup_ind_oth
                         "work_loc_home","work_loc_foreign","work_loc_notfixed","work_loc_workplace","commute_same_sub
                         "commute_same_prov","commute_diff_prov","commute_transp_cardriver","commute_transp_carpass"
                         "commute_transp_walk","commute_transp_bike","commute_transp_other","commute_time_under15",
                         "commute_time_30","commute_time_45","commute_time_over60","commute_start_5am","commute_st
                         "commute_start_8am","commute_start_9am","commute_start_noon"]
dwelling_vars = ["homes_detached_house","homes_semidetached_house","homes_rowhouse","homes_duplex_apt","homes_lowri
                 "homes_highrise_apt","homes_other_stationary","homes_mobile","condo","non_condo"]


# Load Census Profile data
census_data = pd.read_csv('98-401-X2021013_English_CSV_data.csv', encoding="iso-8859-1")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-4-364ce2b10ad5> in <cell line: 2>()
      1 # Load Census Profile data
----> 2 census_data = pd.read_csv('98-401-X2021013_English_CSV_data.csv', encoding="iso-8859-
1")

                              6 frames
/usr/local/lib/python3.10/dist-packages/pandas/io/common.py in get_handle(path_or_buf, mode,
encoding, compression, memory_map, is_text, errors, storage_options)
    854         if ioargs.encoding and "b" not in ioargs.mode:
    855             # Encoding
--> 856             handle = open(
    857                 handle,
    858                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: '98-401-
```

```
# Filter to Ontario FSAs/postal codes
census_data = census_data[census_data['GEO_NAME'].str.match(r'K|L|M|N|P')]

# Drop extra columns
census_data = census_data[["GEO_NAME","CHARACTERISTIC_ID","C1_COUNT_TOTAL","C2_COUNT_MEN+"]]

# Use the number of men for the "num_male" value, otherwise just take the total
census_data['value'] = census_data.apply(lambda row: row['C2_COUNT_MEN+'] if row['CHARACTERISTIC_ID']==8 else

# Select only those rows corresponding to the characteristics we want
census_data = census_data.loc[census_data['CHARACTERISTIC_ID'].isin(var_names.keys())]

# Add in the variable names
census_data['varname'] = census_data.apply(lambda row: var_names[row['CHARACTERISTIC_ID']], axis=1)

# Pivot/transpose
census_data = census_data.pivot(index='GEO_NAME', columns='varname', values='value')



# Load the EV data
ev_data = pd.read_csv("ontario_evs_by_fsa_2023-03-31.csv")
```

```
# Join EV and census data by FSA
census_data = census_data.merge(ev_data, left_on='GEO_NAME', right_on='FSA', how='inner')

# Normalize data for individuals over the population
for var in ppl_vars:
        census_data[var] = census_data[var]/census_data['total_pop']

# Normalize data for homes over dwelling count
for var in dwelling_vars:
        census_data[var] = census_data[var]/census_data['total_dwellings']

# Normalize EV data over population, and calculate per-10,000
census_data['BEV'] = (census_data['BEV']/census_data['total_pop'])*10000
census_data['PHEV'] = (census_data['PHEV']/census_data['total_pop'])*10000
census_data['TotalEV'] = (census_data['TotalEV']/census_data['total_pop'])*10000


# Clean up - reorder the columns
all_vars.insert(0, 'FSA')
all_vars.extend(['BEV','PHEV','TotalEV'])
data = census_data[all_vars]
```

Feature Selection

```
#data = pd.read_csv("newdata.csv")
```

Now that the dataset is assembled, we will examine its structure, and select features from each domain that best correlate with our target variable.

```
df = pd.DataFrame(data)
```

```
df.head()
```

	Unnamed: 0	FSA	total_pop	total_dwellings	pop_density	land_area	num_male	a
0	0	K0A	111626.0	43737.0	NaN	NaN	0.502258	0.(
1	1	K0B	21020.0	9489.0	NaN	NaN	0.504282	0.(
2	2	K0C	52838.0	22714.0	NaN	NaN	0.498978	0.(
3	3	K0E	39649.0	18426.0	NaN	NaN	0.499887	0.(
4	4	K0G	39862.0	19411.0	NaN	NaN	0.496839	0.(

```
df.corr()
```

income_30k	0.142209	0.018681	0.096864
income_40k	0.239515	-0.068326	0.022472
income_50k	0.207168	-0.133492	-0.052951
income_60k	0.029360	-0.192140	-0.114000
income_70k	-0.099302	-0.157246	-0.091124
income_80k	-0.207523	-0.114700	-0.082131
income_90k	-0.183592	-0.094692	-0.086085
income_100k_up	-0.145841	-0.122640	-0.109712
med_total_household_income	-0.244619	0.062834	-0.132781
med_aftertax_household_income	-0.244828	0.084723	-0.124427
indigenous_ppl	0.336357	-0.183989	-0.108537
nonindigenous_ppl	-0.326467	0.219381	0.124441
home_owner	0.076116	-0.122872	-0.097729
home_renter	0.036450	-0.104940	0.080650
home_gov_or_ind_band	0.182273	-0.068847	-0.035424
condo	-0.074746	0.100393	0.161856
non_condo	-0.047327	-0.015677	-0.172645
avg_rooms_home	-0.078504	-0.027388	-0.181258
home_cost_under_30pct	0.209558	-0.353678	-0.124665
home_cost_over_30pct	-0.084205	0.018516	0.162946
can_citizen_ppl	0.051980	-0.196336	-0.179458
non_citizen_ppl	-0.063621	0.241492	0.201821
non_immigrant_ppl	0.157086	-0.300796	-0.186675
immigrant_ppl	-0.175806	0.324770	0.184186
non_perm_res_ppl	0.003372	0.121215	0.162892
no_move_last_yr	0.010188	0.129105	-0.014712
moved_last_yr	-0.033402	-0.089550	0.043140
no_move_5yrs	0.041239	0.105166	-0.003990
moved_last_5yrs	-0.055528	-0.088912	0.024525
school_no_hs	0.254441	0.034664	0.038218
school_hs	0.150044	-0.025609	-0.014352
school_college	0.162289	-0.112132	-0.098342

```
# To find the columns with the highest correlation to TotalEV, we first need to compute the correlat
correlation_matrix = data.corr()

# Now, we extract the correlations of all features with respect to 'TotalEV'
total_ev_correlations = correlation_matrix['TotalEV'].sort_values(ascending=False)
```

```
<ipython-input-202-010ad1a82039>:2: FutureWarning:
```

```
The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False.
```

```
total_ev_correlations
```

```
home_gov_or_ind_band            -0.100223
total_dwellings                 -0.103074
age_55                          -0.127937
home_owner                      -0.129152
no_move_5yrs                    -0.129407
homes_mobile                    -0.131052
occup_cat_health                -0.131106
work_lastyear_parttime          -0.131691
num_male                        -0.133901
homes_duplex_apt                -0.134685
commute_same_div                -0.134852
occup_ind_agr_forest            -0.147151
can_citizen_ppl                 -0.154739
occup_ind_mine_og               -0.161794
homes_lowrise_apt               -0.161830
age_60                          -0.164746
occup_ind_constr                -0.166833
occup_ind_transp_warehs         -0.167163
occup_ind_manuf                 -0.168114
no_move_last_yr                 -0.170206
homes_other_stationary          -0.170483
commute_time_15                 -0.171964
non_condo                       -0.173932
edu_field_agri_res_env          -0.187902
non_immigrant_ppl               -0.187936
occup_cat_natres_agr            -0.193435
occup_ind_other                 -0.214784
commute_same_subdiv             -0.241743
income_50k                      -0.244129
indigenous_ppl                  -0.254281
occup_cat_manuf_util            -0.261751
home_cost_under_30pct           -0.261952
edu_field_health                -0.264745
commute_transp_cardriver        -0.280744
occup_ind_health_socasst        -0.292744
income_10k                      -0.296948
work_loc_notfixed               -0.302315
occup_cat_sales_serv            -0.311535
income_20k                      -0.313368
occup_ind_accom_food_svc        -0.321089
commute_start_5am               -0.325404
occup_ind_retail_trd            -0.326600
commute_start_7am               -0.331429
commute_time_under15            -0.334368
occup_cat_trades_transp         -0.348162
commute_transp_carpass          -0.377344
income_30k                      -0.384509
commute_start_6am               -0.386892
income_40k                      -0.399393
edu_field_pers_protect_transp   -0.407856
school_no_hs                    -0.431955
school_college                  -0.436066
work_loc_workplace              -0.441424
commute_start_noon              -0.464923
school_hs                       -0.504329
pop_density                           NaN
land_area                             NaN
Name: TotalEV, dtype: float64
```

```
selected_features = ["FSA", "med_fulltime_income","income_100k_up", "med_total_household_income", "income_40k",
```

```
# To find the columns with the highest correlation to TotalEV, we first need to compute the correlat:
correlation_matrix = data.corr()

# Now, we extract the correlations of all features with respect to 'TotalEV'
total_ev_correlations = correlation_matrix['TotalEV'].sort_values(ascending=False)

# Next, we'll group the features by their category (e.g., age, commute time, etc.) and select the to|
# from each category based on their absolute correlation values with TotalEV.

# Extracting the prefix of each feature to categorize them
feature_categories = total_ev_correlations.index.str.extract(r'([a-zA-Z_]+)').drop_duplicates().dropna()

# Function to get top N features from each category based on absolute correlation with TotalEV
def get_top_n_features_from_each_category(n=3):
    top_features_per_category = {}

    for category in feature_categories[0]:
        # Filtering the columns by category and then taking the top N
        category_features = total_ev_correlations.filter(regex=f'^{category}', axis=0)
        top_features = category_features.abs().sort_values(ascending=False).head(n).index.tolist()
        top_features_per_category[category] = top_features

    return top_features_per_category

top_features_per_category = get_top_n_features_from_each_category()

top_features_per_category
```

```
 'occup_ind_health_socasst': ['occup_ind_health_socasst'],
 'work_loc_notfixed': ['work_loc_notfixed'],
 'occup_cat_sales_serv': ['occup_cat_sales_serv'],
 'occup_ind_accom_food_svc': ['occup_ind_accom_food_svc'],
 'occup_ind_retail_trd': ['occup_ind_retail_trd'],
 'commute_time_under': ['commute_time_under15'],
 'occup_cat_trades_transp': ['occup_cat_trades_transp'],
 'commute_transp_carpass': ['commute_transp_carpass'],
 'edu_field_pers_protect_transp': ['edu_field_pers_protect_transp'],
 'school_no_hs': ['school_no_hs'],
 'school_college': ['school_college'],
 'work_loc_workplace': ['work_loc_workplace'],
 'commute_start_noon': ['commute_start_noon'],
 'school_hs': ['school_hs'],
 'pop_density': ['pop_density'],
 'land_area': ['land_area']}
```

```
df_new = df[selected_features]
```

```
# To find the columns with the highest correlation to TotalEV, we first need to compute the correlat
correlation_matrix = df_new.corr()

# Now, we extract the correlations of all features with respect to 'TotalEV'
total_ev_correlations = correlation_matrix['TotalEV'].sort_values(ascending=False)
```

```
<ipython-input-207-f27c1cd76d68>:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False.
```

```
total_ev_correlations
```

```
TotalEV                          1.000000
med_fulltime_income              0.671854
income_100k_up                   0.462874
occup_cat_snr_mgmt               0.438262
occup_cat_busfin                 0.402353
work_loc_home                    0.400810
med_total_household_income       0.392518
occup_ind_realestate             0.374130
school_uni_degree                0.373412
edu_field_science                0.352018
edu_field_bus_admin              0.351657
worktype_selfemp                 0.342629
condo                            0.214618
married_ppl                      0.174152
commute_start_9am                0.144460
work_loc_foreign                 0.127200
non_citizen_ppl                  0.097063
can_citizen_ppl                 -0.154739
indigenous_ppl                  -0.254281
edu_field_health                -0.264745
work_loc_notfixed               -0.302315
commute_transp_carpass          -0.377344
commute_start_6am               -0.386892
income_40k                      -0.399393
edu_field_pers_protect_transp   -0.407856
work_loc_workplace              -0.441424
commute_start_noon              -0.464923
school_hs                       -0.504329
Name: TotalEV, dtype: float64
```

```
df_new.to_csv("data.csv")
```

Data Ingestion Pipeline

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
import plotly.express as px
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
```

```
data = pd.read_csv("data.csv")
```

```
pd.set_option('display.max_columns', None)
pd.set_option("display.max_rows", None)
df = pd.DataFrame(data)
df.head(15)
```

	Unnamed: 0	FSA	med_fulltime_income	income_100k_up	med_total_household_incom
0	0	K0A	74500.0	0.127076	115000.
1	1	K0B	56800.0	0.063511	79000.
2	2	K0C	58800.0	0.066429	84000.
3	3	K0E	58000.0	0.063558	83000.
4	4	K0G	64500.0	0.094451	94000.
5	5	K0H	64000.0	0.082900	92000.
6	6	K0J	60000.0	0.071091	76000.
7	7	K0K	58800.0	0.069055	83000.
8	8	K0L	59200.0	0.069489	79000.
9	9	K0M	58800.0	0.074422	80000.
10	10	K1A	NaN	0.075758	73500.
11	11	K1B	65500.0	0.084135	96000.
12	12	K1C	78500.0	0.134417	117000.
13	13	K1E	71000.0	0.104770	110000.
14	14	K1G	68000.0	0.090665	83000.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 518 entries, 0 to 517
Data columns (total 30 columns):
 #   Column                        Non-Null Count  Dtype
```

```
---  ------                       --------------  -----
 0   Unnamed: 0                   518 non-null    int64
 1   FSA                          518 non-null    object
 2   med_fulltime_income          514 non-null    float64
 3   income_100k_up               515 non-null    float64
 4   med_total_household_income   515 non-null    float64
 5   income_40k                   515 non-null    float64
 6   occup_cat_snr_mgmt           515 non-null    float64
 7   occup_cat_busfin             515 non-null    float64
 8   occup_ind_realestate         515 non-null    float64
 9   work_loc_home                515 non-null    float64
 10  work_loc_workplace           515 non-null    float64
 11  work_loc_notfixed            515 non-null    float64
 12  school_uni_degree            515 non-null    float64
 13  school_hs                    515 non-null    float64
 14  commute_start_noon           515 non-null    float64
 15  edu_field_science            515 non-null    float64
 16  edu_field_bus_admin          515 non-null    float64
 17  edu_field_health             515 non-null    float64
 18  edu_field_pers_protect_transp  515 non-null    float64
 19  commute_transp_carpass       515 non-null    float64
 20  commute_start_6am            515 non-null    float64
 21  commute_start_9am            515 non-null    float64
 22  condo                        515 non-null    float64
 23  worktype_selfemp             515 non-null    float64
 24  indigenous_ppl               515 non-null    float64
 25  married_ppl                  515 non-null    float64
 26  work_loc_foreign             515 non-null    float64
 27  can_citizen_ppl              515 non-null    float64
 28  non_citizen_ppl              515 non-null    float64
 29  TotalEV                      518 non-null    float64
dtypes: float64(28), int64(1), object(1)
memory usage: 121.5+ KB
```

```
df = df.convert_dtypes()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 518 entries, 0 to 517
Data columns (total 30 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Unnamed: 0                   518 non-null    Int64
 1   FSA                          518 non-null    string
 2   med_fulltime_income          514 non-null    Int64
 3   income_100k_up               515 non-null    Float64
 4   med_total_household_income   515 non-null    Int64
 5   income_40k                   515 non-null    Float64
 6   occup_cat_snr_mgmt           515 non-null    Float64
 7   occup_cat_busfin             515 non-null    Float64
 8   occup_ind_realestate         515 non-null    Float64
 9   work_loc_home                515 non-null    Float64
 10  work_loc_workplace           515 non-null    Float64
 11  work_loc_notfixed            515 non-null    Float64
 12  school_uni_degree            515 non-null    Float64
 13  school_hs                    515 non-null    Float64
 14  commute_start_noon           515 non-null    Float64
 15  edu_field_science            515 non-null    Float64
 16  edu_field_bus_admin          515 non-null    Float64
 17  edu_field_health             515 non-null    Float64
 18  edu_field_pers_protect_transp  515 non-null    Float64
 19  commute_transp_carpass       515 non-null    Float64
 20  commute_start_6am            515 non-null    Float64
 21  commute_start_9am            515 non-null    Float64
 22  condo                        515 non-null    Float64
 23  worktype_selfemp             515 non-null    Float64
 24  indigenous_ppl               515 non-null    Float64
 25  married_ppl                  515 non-null    Float64
 26  work_loc_foreign             515 non-null    Float64
 27  can_citizen_ppl              515 non-null    Float64
```

```
 28   non_citizen_ppl              515 non-null    Float64
 29   TotalEV                      518 non-null    Float64
dtypes: Float64(26), Int64(3), string(1)
memory usage: 136.2 KB
```

Each forward sortation area has total number of people residing at that region. Now we have features that describe the total number of people who belong to that particular feature.

Missing Values

```
df[df.isna().any(axis=1)]
```

	Unnamed: 0	FSA	med_fulltime_income	income_100k_up	med_total_household_incoı
10	10	K1A	<NA>	0.075758	735(
158	158	L4V	<NA>	<NA>	<N/
175	175	L5S	<NA>	<NA>	<N/
176	176	L5T	<NA>	<NA>	<N/

```
index_label = [10,158,175,176] #We are removing null values here and the TotalEV bias that you can se
```

```
df = df.drop(index_label)
```

```
df = df.drop('Unnamed: 0', axis=1)
```

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

```
FSA                            0
med_fulltime_income            0
income_100k_up                 0
med_total_household_income     0
income_40k                     0
occup_cat_snr_mgmt             0
occup_cat_busfin               0
occup_ind_realestate           0
work_loc_home                  0
work_loc_workplace             0
work_loc_notfixed              0
school_uni_degree              0
school_hs                      0
commute_start_noon             0
edu_field_science              0
edu_field_bus_admin            0
edu_field_health               0
edu_field_pers_protect_transp  0
commute_transp_carpass         0
commute_start_6am              0
commute_start_9am              0
condo                          0
worktype_selfemp               0
indigenous_ppl                 0
married_ppl                    0
work_loc_foreign               0
can_citizen_ppl                0
non_citizen_ppl                0
TotalEV                        0
dtype: int64
```

```
df.loc[0]
```

```
FSA                                    K0A
med_fulltime_income                  74500
income_100k_up                    0.127076
med_total_household_income          115000
income_40k                        0.078879
occup_cat_snr_mgmt                0.009093
occup_cat_busfin                  0.104232
occup_ind_realestate              0.007256
work_loc_home                     0.170659
work_loc_workplace                0.267635
work_loc_notfixed                 0.075341
school_uni_degree                  0.14777
school_hs                         0.131869
commute_start_noon                0.038477
edu_field_science                 0.013841
edu_field_bus_admin                0.06916
edu_field_health                  0.047794
edu_field_pers_protect_transp      0.02598
commute_transp_carpass            0.021276
commute_start_6am                 0.092004
commute_start_9am                 0.026204
condo                             0.026865
worktype_selfemp                  0.082911
indigenous_ppl                    0.040895
married_ppl                       0.554754
work_loc_foreign                  0.000806
can_citizen_ppl                   0.971234
non_citizen_ppl                   0.015677
TotalEV                          88.420261
Name: 0, dtype: object
```

```
df.head()
```

	FSA	med_fulltime_income	income_100k_up	med_total_household_income	income_4(
0	K0A	74500	0.127076	115000	0.0788
1	K0B	56800	0.063511	79000	0.0980
2	K0C	58800	0.066429	84000	0.0961
3	K0E	58000	0.063558	83000	0.1025
4	K0G	64500	0.094451	94000	0.0925

Continuous Features report

```
def build_continuous_features_report(data_df):
    stats = {
        "Count": len,
        "Miss %": lambda df: df.isna().sum() / len(df) * 100,
        "Card.": lambda df: df.nunique(),
        "Min": lambda df: df.min(),
        "1st Qrt.": lambda df: df.quantile(0.25),
        "Mean": lambda df: df.mean(),
        "Median": lambda df: df.median(),
        "3rd Qrt": lambda df: df.quantile(0.75),
        "Max": lambda df: df.max(),
        "Std. Dev.": lambda df: df.std(),
    }

    contin_feat_names = data_df.select_dtypes("number").columns
    continuous_data_df = data_df[contin_feat_names]

    report_df = pd.DataFrame(index=contin_feat_names, columns=stats.keys())

    for stat_name, fn in stats.items():
        # NOTE: ignore warnings for empty features
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            report_df[stat_name] = fn(continuous_data_df)

    return report_df


build_continuous_features_report(df)
```

	Count	Miss %	Card.	Min	1st Qrt.	Me
med_fulltime_income	514	0.0	97	44400.000000	58800.0	68139.1050
income_100k_up	514	0.0	514	0.012731	0.061412	0.1001
med_total_household_income	514	0.0	121	45200.000000	77000.0	94259.1439
income_40k	514	0.0	514	0.047174	0.0734	0.0843
occup_cat_snr_mgmt	514	0.0	507	0.000000	0.003388	0.0072
occup_cat_busfin	514	0.0	514	0.032976	0.069007	0.0913
occup_ind_realestate	514	0.0	513	0.000000	0.006928	0.0104
work_loc_home	514	0.0	514	0.023318	0.079552	0.1326
work_loc_workplace	514	0.0	513	0.167441	0.240323	0.2681
work_loc_notfixed	514	0.0	514	0.017125	0.043569	0.0528
school_uni_degree	514	0.0	514	0.041984	0.10895	0.1897
school_hs	514	0.0	513	0.015504	0.1033	0.1252
commute_start_noon	514	0.0	514	0.011377	0.039829	0.0512
edu_field_science	514	0.0	514	0.002772	0.009196	0.0150
edu_field_bus_admin	514	0.0	514	0.027260	0.056779	0.0786
edu_field_health	514	0.0	514	0.022753	0.044236	0.0508
edu_field_pers_protect_transp	514	0.0	511	0.000000	0.014907	0.0196
commute_transp_carpass	514	0.0	513	0.005306	0.019519	0.0235
commute_start_6am	514	0.0	514	0.000000	0.04431	0.0567
commute_start_9am	514	0.0	514	0.020504	0.035225	0.0430
condo	514	0.0	498	0.000000	0.032001	0.1240
worktype_selfemp	514	0.0	514	0.024189	0.05555	0.0752
indigenous_ppl	514	0.0	510	0.000000	0.009151	0.0365
married_ppl	514	0.0	514	0.319341	0.446095	0.4781
work_loc_foreign	514	0.0	440	0.000000	0.00078	0.0021
can_citizen_ppl	514	0.0	514	0.626602	0.854411	0.8981
non_citizen_ppl	514	0.0	513	0.000000	0.025141	0.0863
TotalEV	514	0.0	514	4.320276	40.721371	78.5657

We see the mean of the TotalEV is 78.56 but ther maximum value is 542.63. We might have to explore on this later

Categorical Features Report

```
df.describe(exclude=['number'])
```

	FSA
count	514
unique	514
top	K0A
freq	1

This confirms us that there are no duplicates in the forward sortation area

```
def build_categorical_features_report(data_df):

    def _mode(df):
        return df.apply(lambda ft: ft.mode().to_list())

    def _mode_freq(df):
        return df.apply(lambda ft: ft.value_counts()[ft.mode()].sum())

    def _second_mode(df):
        return df.apply(lambda ft: ft[~ft.isin(ft.mode())].mode().to_list())

    def _second_mode_freq(df):
        return df.apply(
            lambda ft: ft[~ft.isin(ft.mode())]
            .value_counts()[ft[~ft.isin(ft.mode())].mode()]
            .sum()
        )

    stats = {
        "Count": len,
        "Miss %": lambda df: df.isna().sum() / len(df) * 100,
        "Card.": lambda df: df.nunique(),
        "Mode": _mode,
        "Mode Freq": _mode_freq,
        "Mode %": lambda df: _mode_freq(df) / len(df) * 100,
        "2nd Mode": _second_mode,
        "2nd Mode Freq": _second_mode_freq,
        "2nd Mode %": lambda df: _second_mode_freq(df) / len(df) * 100,
    }

    cat_feat_names = data_df.select_dtypes(exclude="number").columns
    continuous_data_df = data_df[cat_feat_names]

    report_df = pd.DataFrame(index=cat_feat_names, columns=stats.keys())

    for stat_name, fn in stats.items():
        # NOTE: ignore warnings for empty features
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            report_df[stat_name] = fn(continuous_data_df)

    return report_df
```

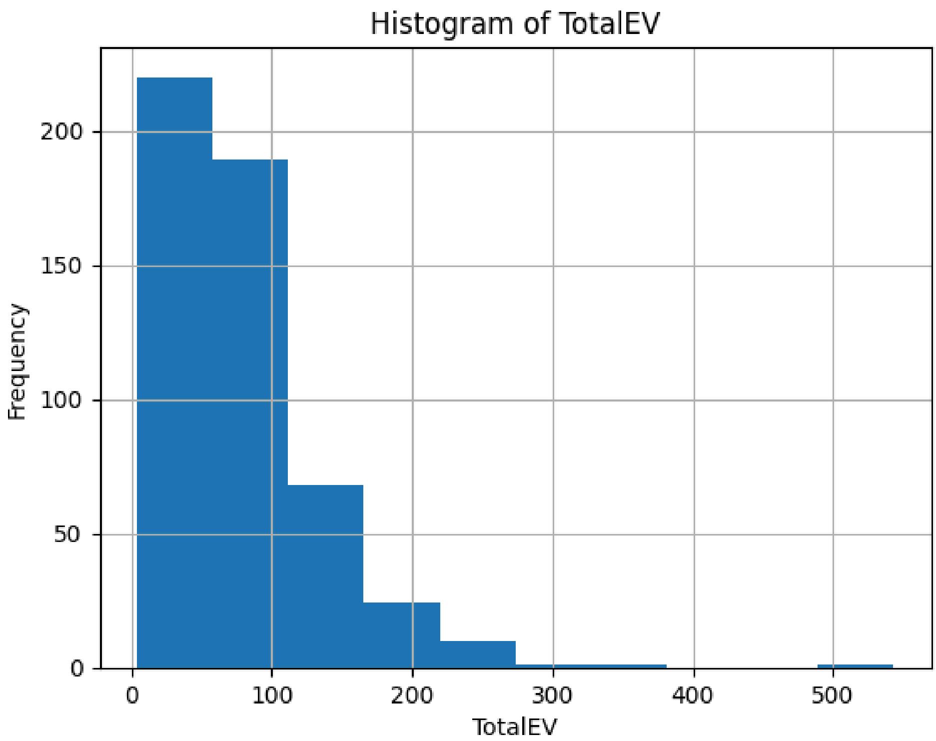
```
build_categorical_features_report(df)
```

	Count	Miss %	Card.	Mode	Mode Freq	Mode %	2nd Mode	2nd Mode Freq	2nd Mode %
FSA	514	0.0	514	NaN	514	100.0	NaN	0	0.0

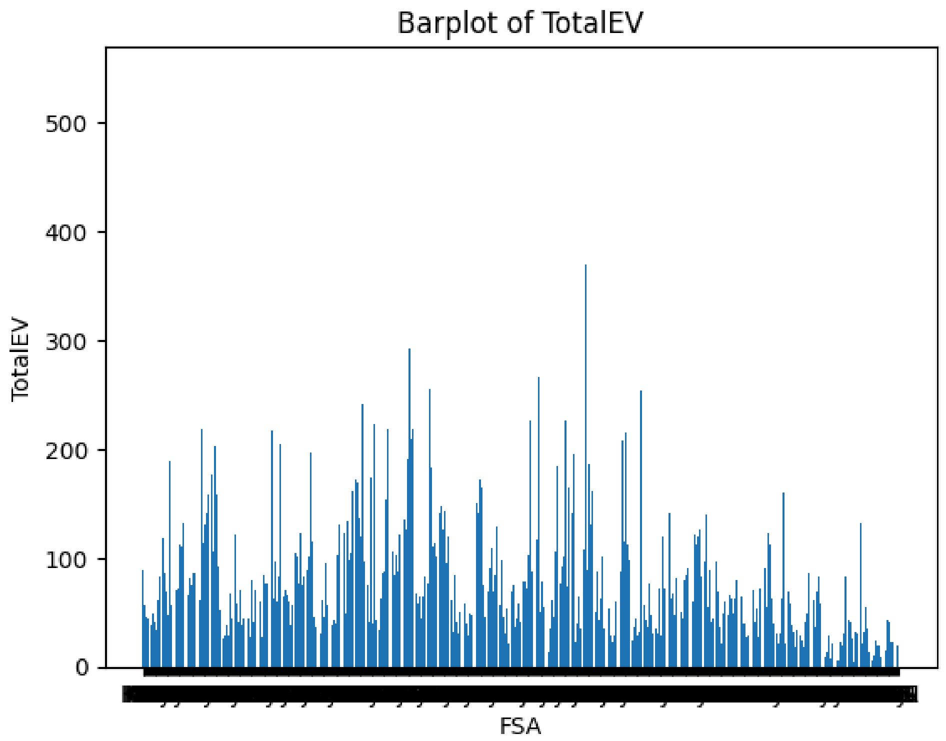
Visualizing the Features

As mentioned previously in the continuous feature report section, we will look how the TotalEV feature is dispersed

```
df['TotalEV'].hist(bins=10)
plt.xlabel('TotalEV')
plt.ylabel('Frequency')
plt.title('Histogram of TotalEV')
plt.show()
```




```
plt.bar(df["FSA"],df["TotalEV"])
plt.xlabel('FSA')
plt.ylabel('TotalEV')
plt.title('Barplot of TotalEV')
plt.show()
```

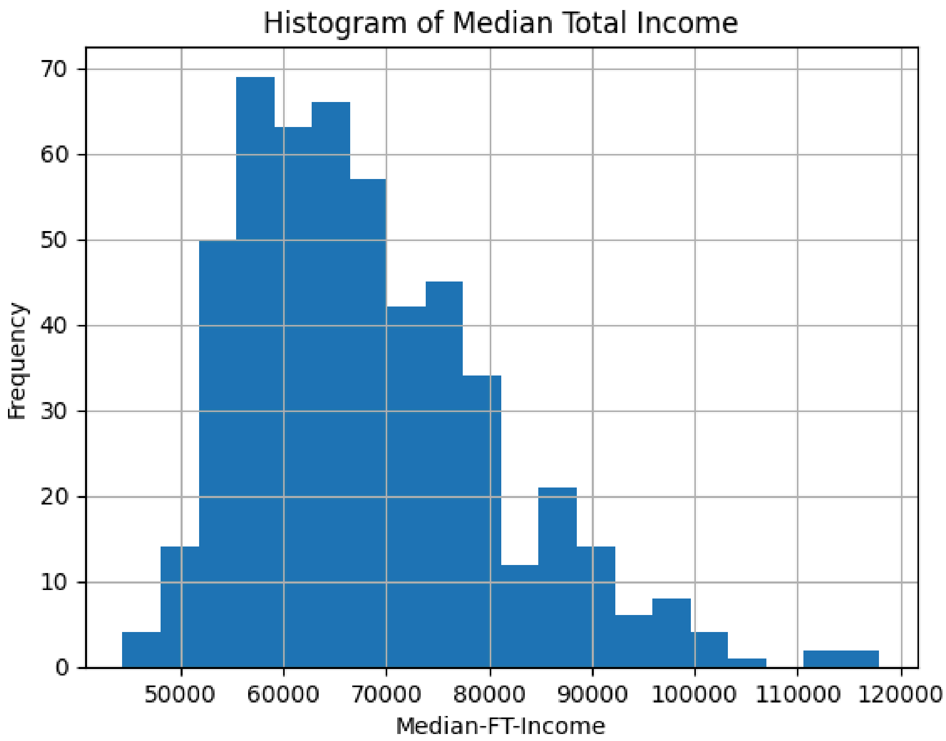


The way the graph has displayed the distribution of values is not bad. The totalEV frequency is decreasing as the number of TotalEV is increasing

```
df.head()
```

	FSA	med_fulltime_income	income_100k_up	med_total_household_income	income_4(
0	K0A	74500	0.127076	115000	0.0788
1	K0B	56800	0.063511	79000	0.0980
2	K0C	58800	0.066429	84000	0.0961
3	K0E	58000	0.063558	83000	0.1025
4	K0G	64500	0.094451	94000	0.0925

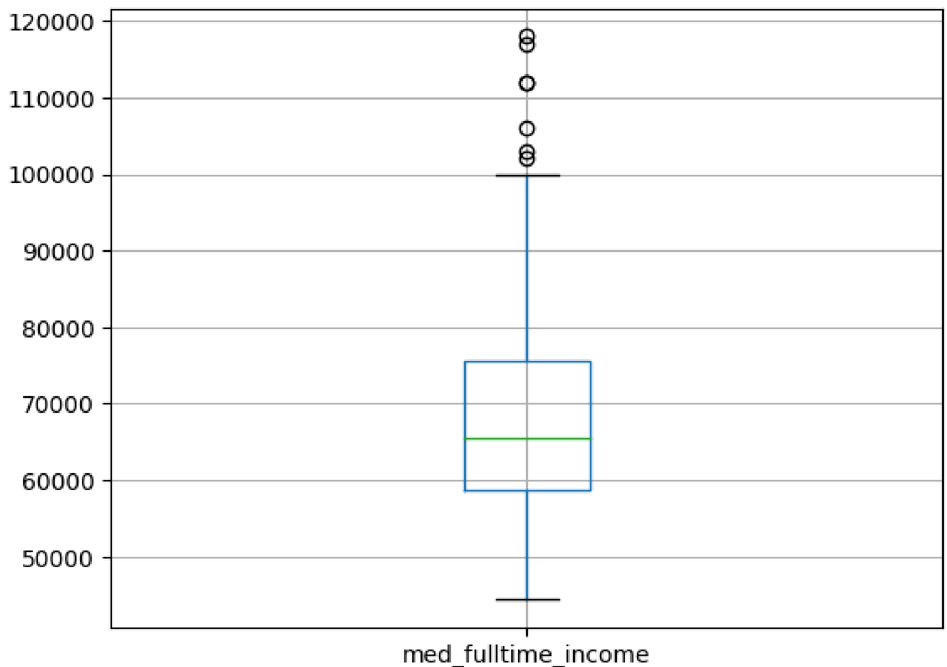
```
df['med_fulltime_income'].hist(bins=20)
plt.xlabel('Median-FT-Income')
plt.ylabel('Frequency')
plt.title('Histogram of Median Total Income')
plt.show()
```



- We see a graphical representation being a little right skewed but not much. The graph is bi-modal, meaning there are two distinct peaks.
- The bars in the right most end have some exceptionally high values. We must investigate if they are outliers or a data entry error

```
df.boxplot(column=['med_fulltime_income'])
```

<Axes: >



Let's try to find out where the median total income is this high. If it's a city or a

```
df.sort_values(by='med_fulltime_income', ascending=False)
```

	FSA	med_fulltime_income	income_100k_up	med_total_household_income	in
307	M5M	118000	0.265738	164000	
41	K2R	117000	0.234438	164000	
330	M8X	112000	0.28238	139000	
291	M4N	112000	0.272138	146000	
285	M4G	106000	0.257424	130000	
185	L6J	103000	0.225635	158000	
293	M4R	102000	0.240574	107000	
19	K1M	100000	0.278681	128000	
295	M4T	100000	0.281165	114000	
284	M4E	98000	0.217289	112000	
45	K2W	98000	0.212206	159000	
297	M4W	97000	0.279158	104000	
43	K2T	97000	0.200489	141000	
327	M6S	97000	0.208205	108000	
30	K2A	97000	0.213919	105000	
273	M3B	96000	0.21118	121000	
303	M5E	96000	0.26785	98000	
106	L1M	95000	0.183657	154000	
188	L6M	95000	0.180435	143000	
168	L5H	94000	0.213683	139000	
23	K1S	93000	0.229069	106000	
289	M4L	93000	0.186852	101000	
187	L6L	93000	0.183427	122000	
50	K4M	92000	0.208623	146000	
51	K4P	92000	0.198124	155000	
439	N7X	92000	0.230292	142000	
305	M5H	92000	0.226293	90000	
42	K2S	91000	0.181606	137000	
129	L2W	91000	0.137825	110000	
481	P1C	91000	0.201721	153000	

We see that M5M has the highest number of income. It is evident because, M5M represents Toronto, a pretty populous and developed city in terms of number of civilians. Hence we shall not consider this as an outlier

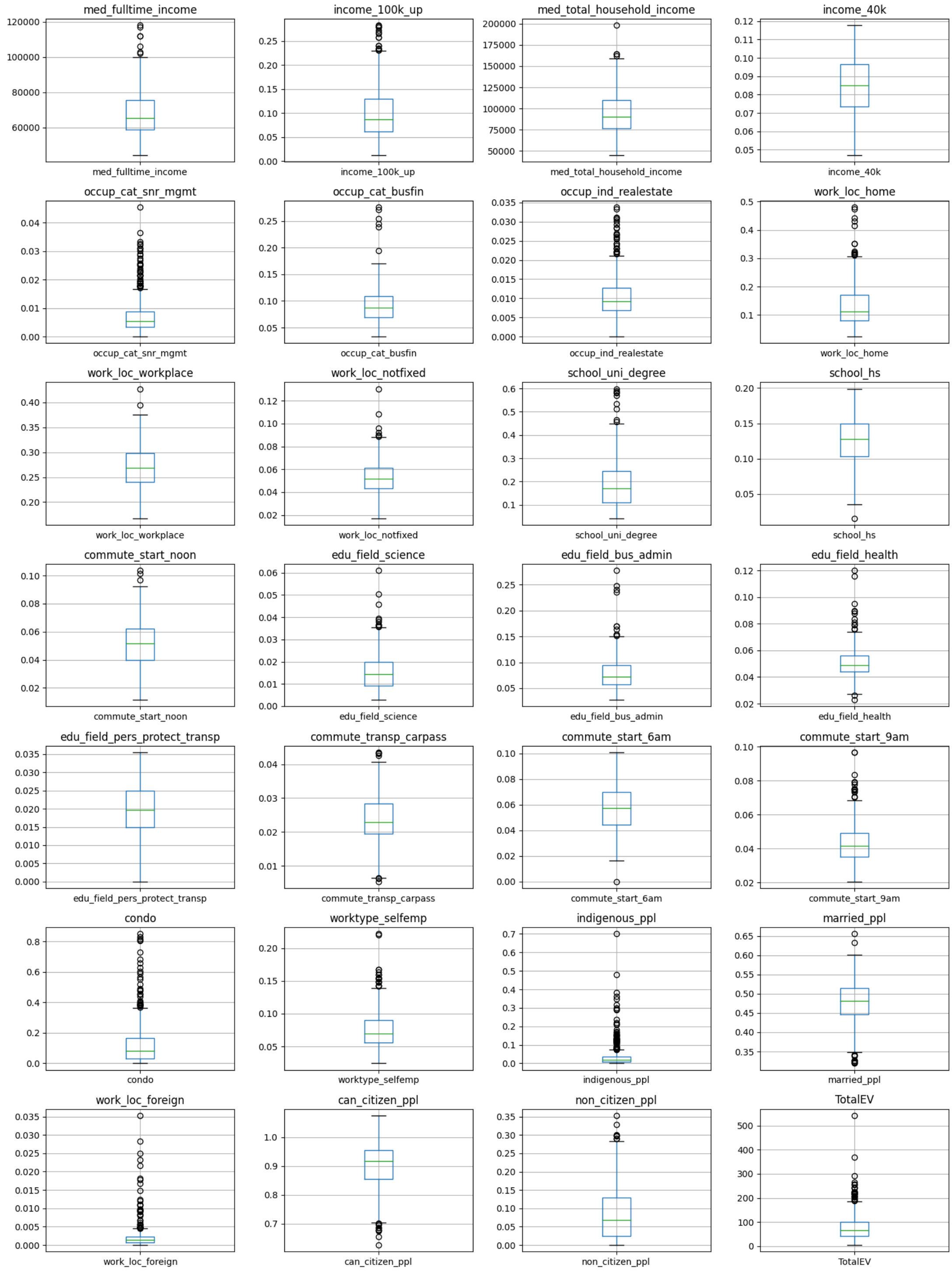
```
#Selecting numeric columns to visualize outliers
numeric_columns = df.select_dtypes(include=["int", "float"]).columns
numeric_columns
```

```
Index(['med_fulltime_income', 'income_100k_up', 'med_total_household_income',
       'income_40k', 'occup_cat_snr_mgmt', 'occup_cat_busfin',
       'occup_ind_realestate', 'work_loc_home', 'work_loc_workplace',
       'work_loc_notfixed', 'school_uni_degree', 'school_hs',
       'commute_start_noon', 'edu_field_science', 'edu_field_bus_admin',
       'edu_field_health', 'edu_field_pers_protect_transp',
       'commute_transp_carpass', 'commute_start_6am', 'commute_start_9am',
       'condo', 'worktype_selfemp', 'indigenous_ppl', 'married_ppl',
       'work_loc_foreign', 'can_citizen_ppl', 'non_citizen_ppl', 'TotalEV'],
      dtype='object')
```

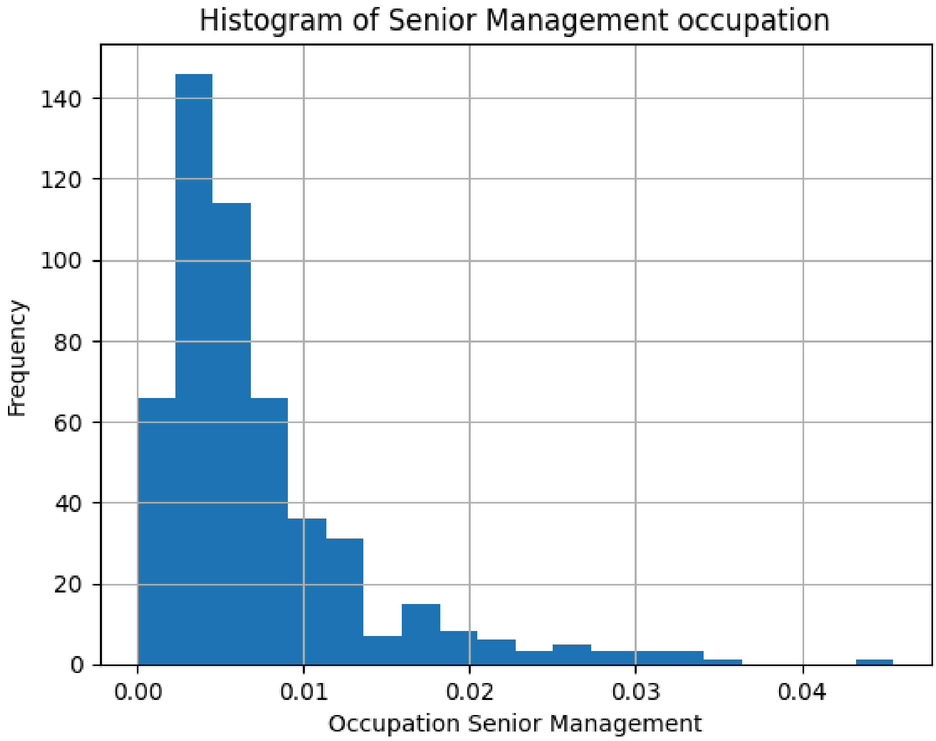
```
len(numeric_columns)
```

```
28
```

```
fig, axes = plt.subplots(nrows=7, ncols=4, figsize=(15, 20))
axes = axes.flatten()

for i, col in enumerate(numeric_columns):
    ax = axes[i]
    df.boxplot(column=col, ax=ax)
    ax.set_title(col)

plt.tight_layout()
plt.show()
```

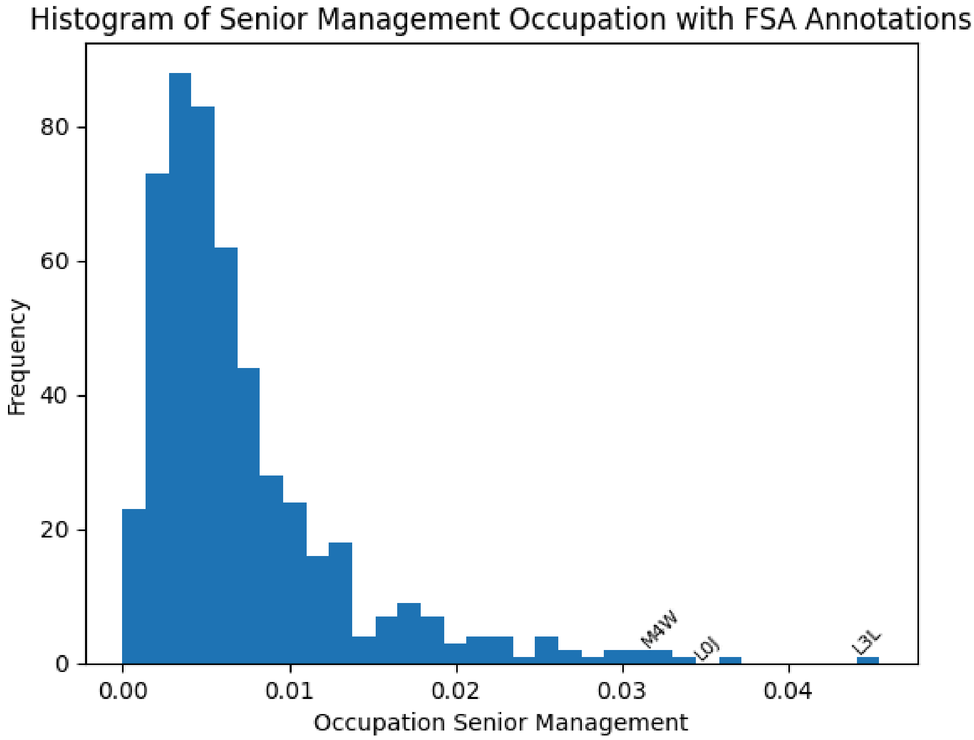

med_fulltime_income
income_100k_up
med_total_household_income
income_40k
occup_cat_snr_mgmt
occup_cat_busfin
occup_ind_realestate
work_loc_home
work_loc_workplace
work_loc_notfixed
school_uni_degree
school_hs
commute_start_noon
edu_field_science
edu_field_bus_admin
edu_field_health
edu_field_pers_protect_transp
commute_transp_carpass
commute_start_6am
commute_start_9am
condo
worktype_selfemp
indigenous_ppl
married_ppl
work_loc_foreign
can_citizen_ppl
non_citizen_ppl
TotalEV

```
df['occup_cat_snr_mgmt'].hist(bins=20)
plt.xlabel('Occupation Senior Management')
plt.ylabel('Frequency')
plt.title('Histogram of Senior Management occupation')
plt.show()
```



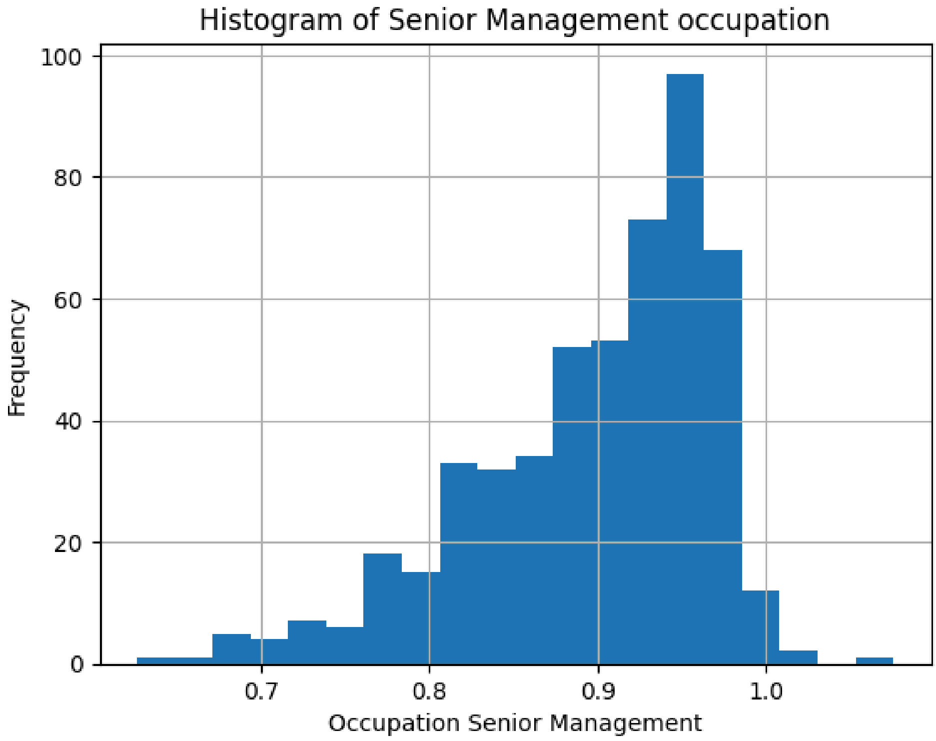

```
# Step 1: Determine the Bin Ranges
last_three_values = sorted(df['occup_cat_snr_mgmt'].unique())[-3:]
print(last_three_values)

# Step 2: Create a Histogram and store the bin information
n, bins, patches = plt.hist(df['occup_cat_snr_mgmt'], bins='auto')  # 'auto' can be replaced with specif

# Step 3: Annotate the Histogram
# Find the center of each of the last five bins to place the text
for value in last_three_values:
        # Assuming that the value aligns exactly with a bin edge; otherwise, find the nearest bin
        bin_index = (np.abs(bins - value)).argmin() - 1  # Get the index of the bin to the left of
        bin_center = (bins[bin_index] + bins[bin_index + 1]) / 2
        bin_height = n[bin_index]

        # Get the corresponding FSA names for this bin
        fsa_names = df[df['occup_cat_snr_mgmt'] == value]['FSA'].unique()
        fsa_names_str = ", ".join(fsa_names)  # Concatenate FSA names into a single string

        # Annotate the histogram with FSA names
        plt.text(bin_center, bin_height, fsa_names_str, ha='center', va='bottom', fontsize=8, rotation=45)

plt.title('Histogram of Senior Management Occupation with FSA Annotations')
plt.xlabel('Occupation Senior Management')
plt.ylabel('Frequency')
plt.show()
```

[0.033342050209205, 0.0364004044489383, 0.0455062571103526]



1. **Distribution Shape**: The distribution of the 'Occupation Senior Management' variable appears to be right-skewed, with a majority of the data concentrated on the left side of the histogram.
2. **Common Values**: The highest frequency of Senior Management Occupation values is in the lower range (close to 0), indicating that higher percentages in this category are less common.
3. **Tail of the Distribution**: There is a long tail extending towards the right, which indicates that there are relatively few FSAs with a higher proportion of individuals in senior management roles.
4. **FSA Annotations**: Annotations on the histogram indicate specific FSAs associated with the higher end of the distribution. These FSAs are likely those with the highest proportion of individuals in senior management occupations, and they stand out from the general trend.
5. **Outliers**: The presence of FSAs with a higher proportion at the tail end could also indicate potential outliers, or it could simply reflect areas with an unusually high concentration of senior management occupations.
6. **Analysis Focus**: The annotations indicate specific areas of interest that may warrant further investigation. For example, why do these FSAs have a higher proportion of senior management occupations? This could be related to various factors like the presence of corporate headquarters, higher economic activity, or a different industrial mix compared to other FSAs.

```
df['can_citizen_ppl'].hist(bins=20)
plt.xlabel('Occupation Senior Management')
plt.ylabel('Frequency')
plt.title('Histogram of Senior Management occupation')
plt.show()
```



Categorizing the TotalEV using Quantiles and Clustering Similar Features with TotalEV

```
# Calculate the percentiles
#low_threshold = df['TotalEV'].quantile(0.33)
#high_threshold = df['TotalEV'].quantile(0.66)
low_threshold = 50
high_threshold = 100

# Create the categories
df['TotalEV_Category'] = pd.cut(df['TotalEV'], bins=[-float('inf'), low_threshold, high_threshold, float('inf')]
```

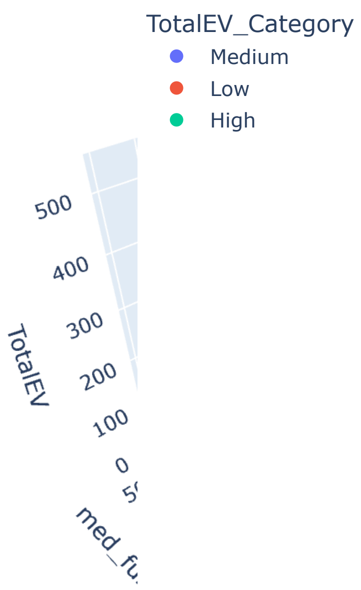
```
df['TotalEV_Category'].unique()
```

```
['Medium', 'Low', 'High']
Categories (3, object): ['Low' < 'Medium' < 'High']
```

```
df.head()
```

	FSA	med_fulltime_income	income_100k_up	med_total_household_income	income_4(
0	K0A	74500	0.127076	115000	0.0788
1	K0B	56800	0.063511	79000	0.0980
2	K0C	58800	0.066429	84000	0.0961
3	K0E	58000	0.063558	83000	0.1025
4	K0G	64500	0.094451	94000	0.0925

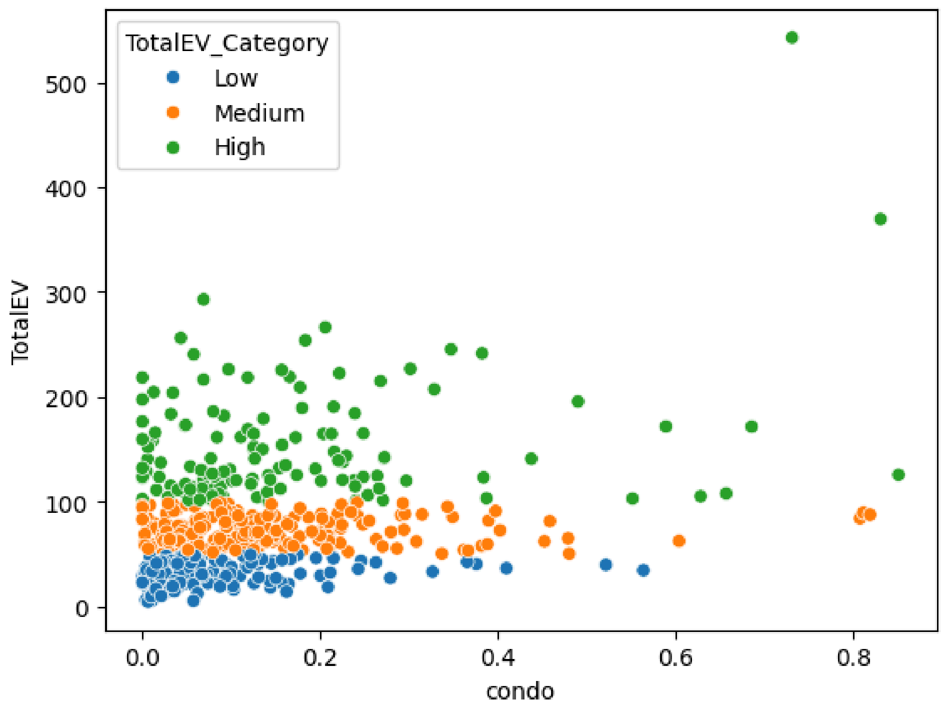
```
fig = px.scatter_3d(df, z='TotalEV', y='med_fulltime_income', x='med_total_household_income',color='TotalEV_Cate
fig.show()
```



- **Correlation**: There seems to be a positive correlation between `med_total_household_income` and `TotalEV`, as well as between `med_fulltime_income` and `TotalEV`. This suggests that as median incomes increase, the `TotalEV` also tends to increase.
- **Income Disparity**: The spread along the `med_fulltime_income` axis appears to be less pronounced than along the `med_total_household_income`, which might indicate less variability in full-time income as compared to household income.
- **Data Density**: The data points are more concentrated in the lower to middle income brackets for both `med_total_household_income` and `med_fulltime_income`, which suggests that there are fewer data points (FSAs or regions) with very high incomes.
- **Outliers**: There are some outliers, particularly visible in the `TotalEV` dimension (points that are much higher than the main cluster of data), indicating some FSAs or regions with significantly higher values of `TotalEV`.

```
sns.scatterplot(x=df["condo"], y=df["TotalEV"], hue=df['TotalEV_Category'])
```

```
<Axes: xlabel='condo', ylabel='TotalEV'>
```



From this graph, several inferences can be made:

1. **Positive Correlation**: There appears to be a positive correlation between the 'condo' metric and 'TotalEV'. As the 'condo' metric increases, 'TotalEV' also tends to increase.
2. **Category Distribution**: The 'TotalEV_Category' shows that most of the low and medium categories are clustered at the lower end of the 'condo' metric. In contrast, the high category is more spread out and appears more frequently at higher 'condo' values.
3. **Concentration of Categories**: There's a higher concentration of low and medium 'TotalEV' categories in the lower range of the 'condo' metric. The high 'TotalEV' category seems to be more scattered and less dense, which might suggest that high 'TotalEV' values occur less frequently or are less tightly correlated with the 'condo' metric.
4. **Data Density**: The majority of the data points are clustered in the lower left corner of the plot, indicating a high density of lower 'TotalEV' values associated with lower 'condo' metric values.
5. **Outliers**: A few data points in the high 'TotalEV' category stand out significantly above the general trend. These could be outliers or indicate specific instances where the 'TotalEV' is exceptionally high relative to the 'condo' metric.
6. **Potential Ceiling Effect**: There might be a ceiling effect visible for the 'condo' metric, as there are a number of high 'TotalEV' points that line up along specific 'condo' values.

```
sns.scatterplot(x=df["work_loc_home"], y=df["TotalEV"], hue=df['TotalEV_Category'])
```