Import liberaries

In [3]:
```
!pip install ISLP
```

```
Collecting ISLP
  Downloading ISLP-0.4.0-py3-none-any.whl.metadata (7.0 kB)
Requirement already satisfied: numpy>=1.7.1 in /usr/local/lib/python3.11/dist-pac
kages (from ISLP) (2.0.2)
Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.11/dist-packa
ges (from ISLP) (1.14.1)
Requirement already satisfied: pandas>=0.20 in /usr/local/lib/python3.11/dist-pac
kages (from ISLP) (2.2.2)
Requirement already satisfied: lxml in /usr/local/lib/python3.11/dist-packages (f
rom ISLP) (5.3.2)
Requirement already satisfied: scikit-learn>=1.2 in /usr/local/lib/python3.11/dis
t-packages (from ISLP) (1.6.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages
(from ISLP) (1.4.2)
Requirement already satisfied: statsmodels>=0.13 in /usr/local/lib/python3.11/dis
t-packages (from ISLP) (0.14.4)
Collecting lifelines (from ISLP)
  Downloading lifelines-0.30.0-py3-none-any.whl.metadata (3.2 kB)
Collecting pygam (from ISLP)
  Downloading pygam-0.9.1-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages
(from ISLP) (2.6.0+cu124)
Collecting pytorch-lightning (from ISLP)
  Downloading pytorch_lightning-2.5.1-py3-none-any.whl.metadata (20 kB)
Collecting torchmetrics (from ISLP)
  Downloading torchmetrics-1.7.1-py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.1
1/dist-packages (from pandas>=0.20->ISLP) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pac
kages (from pandas>=0.20->ISLP) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-p
ackages (from pandas>=0.20->ISLP) (2025.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/
dist-packages (from scikit-learn>=1.2->ISLP) (3.6.0)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-pac
kages (from statsmodels>=0.13->ISLP) (1.0.1)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-
packages (from statsmodels>=0.13->ISLP) (24.2)
Requirement already satisfied: matplotlib>=3.0 in /usr/local/lib/python3.11/dist-
packages (from lifelines->ISLP) (3.10.0)
Requirement already satisfied: autograd>=1.5 in /usr/local/lib/python3.11/dist-pa
ckages (from lifelines->ISLP) (1.7.0)
Collecting autograd-gamma>=0.3 (from lifelines->ISLP)
  Downloading autograd-gamma-0.5.0.tar.gz (4.0 kB)
  Preparing metadata (setup.py) ... done
Collecting formulaic>=0.2.2 (from lifelines->ISLP)
  Downloading formulaic-1.1.1-py3-none-any.whl.metadata (6.9 kB)
Requirement already satisfied: progressbar2<5.0.0,>=4.2.0 in /usr/local/lib/pytho
n3.11/dist-packages (from pygam->ISLP) (4.5.0)
Collecting scipy>=0.9 (from ISLP)
  Downloading scipy-1.11.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (60 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.4/60.4 kB 4.9 MB/s eta 0:00:00
Collecting numpy>=1.7.1 (from ISLP)
  Downloading numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 4.2 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.57.0 in /usr/local/lib/python3.11/dist-pac
kages (from pytorch-lightning->ISLP) (4.67.1)
Requirement already satisfied: PyYAML>=5.4 in /usr/local/lib/python3.11/dist-pack
```

```
ages (from pytorch-lightning->ISLP) (6.0.2)
Requirement already satisfied: fsspec>=2022.5.0 in /usr/local/lib/python3.11/dist
-packages (from fsspec[http]>=2022.5.0->pytorch-lightning->ISLP) (2025.3.2)
Requirement already satisfied: typing-extensions>=4.4.0 in /usr/local/lib/python
3.11/dist-packages (from pytorch-lightning->ISLP) (4.13.2)
Collecting lightning-utilities>=0.10.0 (from pytorch-lightning->ISLP)
  Downloading lightning_utilities-0.14.3-py3-none-any.whl.metadata (5.6 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-package
s (from torch->ISLP) (3.18.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-package
s (from torch->ISLP) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages
(from torch->ISLP) (3.1.6)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch->ISLP)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.m
etadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch->ISLP)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.wh
l.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch->ISLP)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.m
etadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch->ISLP)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metada
ta (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch->ISLP)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.metad
ata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch->ISLP)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.metada
ta (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch->ISLP)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl.met
adata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch->ISLP)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl.met
adata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch->ISLP)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl.m
etadata (1.6 kB)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/py
thon3.11/dist-packages (from torch->ISLP) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python
3.11/dist-packages (from torch->ISLP) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/pytho
n3.11/dist-packages (from torch->ISLP) (12.4.127)
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch->ISLP)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.me
tadata (1.5 kB)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-pa
ckages (from torch->ISLP) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-pa
ckages (from torch->ISLP) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/di
st-packages (from sympy==1.13.1->torch->ISLP) (1.3.0)
Collecting interface-meta>=1.2.0 (from formulaic>=0.2.2->lifelines->ISLP)
  Downloading interface_meta-1.3.0-py3-none-any.whl.metadata (6.7 kB)
Requirement already satisfied: wrapt>=1.0 in /usr/local/lib/python3.11/dist-packa
ges (from formulaic>=0.2.2->lifelines->ISLP) (1.17.2)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/pytho
```

```
n3.11/dist-packages (from fsspec[http]>=2022.5.0->pytorch-lightning->ISLP) (3.11.
15)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packa
ges (from lightning-utilities>=0.10.0->pytorch-lightning->ISLP) (75.2.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist
-packages (from matplotlib>=3.0->lifelines->ISLP) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-pac
kages (from matplotlib>=3.0->lifelines->ISLP) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dis
t-packages (from matplotlib>=3.0->lifelines->ISLP) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dis
t-packages (from matplotlib>=3.0->lifelines->ISLP) (1.4.8)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packag
es (from matplotlib>=3.0->lifelines->ISLP) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist
-packages (from matplotlib>=3.0->lifelines->ISLP) (3.2.3)
Requirement already satisfied: python-utils>=3.8.1 in /usr/local/lib/python3.11/d
ist-packages (from progressbar2<5.0.0,>=4.2.0->pygam->ISLP) (3.9.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-package
s (from python-dateutil>=2.8.2->pandas>=0.20->ISLP) (1.17.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-
packages (from jinja2->torch->ISLP) (3.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.
11/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->pytorc
h-lightning->ISLP) (2.6.1)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist
-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->pytorch-light
ning->ISLP) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-pa
ckages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->pytorch-lightnin
g->ISLP) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dis
t-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->pytorch-ligh
tning->ISLP) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/d
ist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->pytorch-li
ghtning->ISLP) (6.4.3)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist
-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->pytorch-light
ning->ISLP) (0.3.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dis
t-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->pytorch-ligh
tning->ISLP) (1.19.0)
Requirement already satisfied: idna>=2.0 in /usr/local/lib/python3.11/dist-packag
es (from yarl<2.0,>=1.17.0->aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->p
ytorch-lightning->ISLP) (3.10)
Downloading ISLP-0.4.0-py3-none-any.whl (3.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.6/3.6 MB 57.9 MB/s eta 0:00:00
Downloading lifelines-0.30.0-py3-none-any.whl (349 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 349.3/349.3 kB 26.1 MB/s eta 0:00:00
Downloading pygam-0.9.1-py3-none-any.whl (522 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 522.0/522.0 kB 31.6 MB/s eta 0:00:00
Downloading scipy-1.11.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.w
hl (36.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 36.4/36.4 MB 37.3 MB/s eta 0:00:00
Downloading numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.w
hl (18.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 78.0 MB/s eta 0:00:00
Downloading pytorch_lightning-2.5.1-py3-none-any.whl (822 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 823.0/823.0 kB 46.9 MB/s eta 0:00:00
```

```
Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl (363.4
MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.0 MB/s eta 0:00:00
Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (1
3.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 112.4 MB/s eta 0:00:00
Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (2
4.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 83.8 MB/s eta 0:00:00
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl
(883 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 52.2 MB/s eta 0:00:00
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 M
B)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.1 MB/s eta 0:00:00
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 M
B)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.6 MB/s eta 0:00:00
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3
MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 14.4 MB/s eta 0:00:00
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.
9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.7 MB/s eta 0:00:00
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (20
7.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 5.7 MB/s eta 0:00:00
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.
1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 59.8 MB/s eta 0:00:00
Downloading torchmetrics-1.7.1-py3-none-any.whl (961 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 961.5/961.5 kB 40.6 MB/s eta 0:00:00
Downloading formulaic-1.1.1-py3-none-any.whl (115 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.7/115.7 kB 10.1 MB/s eta 0:00:00
Downloading lightning_utilities-0.14.3-py3-none-any.whl (28 kB)
Downloading interface_meta-1.3.0-py3-none-any.whl (14 kB)
Building wheels for collected packages: autograd-gamma
  Building wheel for autograd-gamma (setup.py) ... done
  Created wheel for autograd-gamma: filename=autograd_gamma-0.5.0-py3-none-any.wh
l size=4030 sha256=687ef96a82cc6627e2a01916581b7bf18a40c7113a782e7491a60d676944af
53
  Stored in directory: /root/.cache/pip/wheels/8b/67/f4/2caaae2146198dcb824f31a30
3833b07b14a5ec863fb3acd7b
Successfully built autograd-gamma
Installing collected packages: nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-
cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-c
u12, nvidia-cublas-cu12, numpy, lightning-utilities, interface-meta, scipy, nvidi
a-cusparse-cu12, nvidia-cudnn-cu12, pygam, nvidia-cusolver-cu12, formulaic, autog
rad-gamma, lifelines, torchmetrics, pytorch-lightning, ISLP
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
```

```
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
  Attempting uninstall: scipy
    Found existing installation: scipy 1.14.1
    Uninstalling scipy-1.14.1:
      Successfully uninstalled scipy-1.14.1
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
ERROR: pip's dependency resolver does not currently take into account all the pac
kages that are installed. This behaviour is the source of the following dependenc
y conflicts.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.26.4 which is inco
mpatible.
Successfully installed ISLP-0.4.0 autograd-gamma-0.5.0 formulaic-1.1.1 interface-
meta-1.3.0 lifelines-0.30.0 lightning-utilities-0.14.3 numpy-1.26.4 nvidia-cublas
-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nv
idia-cuda-runtime-cu12-12.4.127 nvidia-cudnn-cu12-9.1.0.70 nvidia-cufft-cu12-11.
2.1.3 nvidia-curand-cu12-10.3.5.147 nvidia-cusolver-cu12-11.6.1.9 nvidia-cusparse
-cu12-12.3.1.170 nvidia-nvjitlink-cu12-12.4.127 pygam-0.9.1 pytorch-lightning-2.
5.1 scipy-1.11.4 torchmetrics-1.7.1
```

In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from ISLP.models import (ModelSpec as MS,summarize)
import itertools
from prophet import Prophet
```

Import dataset

In [6]:
```
filepath = '/content/drive/MyDrive/Colab Notebooks/Projects/Personal/Real estate
```

In [7]:
```
df = pd.read_csv(filepath)
```

```
<ipython-input-7-7248a6640290>:1: DtypeWarning: Columns (8,9,10,11,12) have mixed
types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv(filepath)
```

In [8]:
```
df.shape
```

Out[8]:
```
(1097629, 14)
```

In [9]:
```
#df.to_excel("Real_Estate_Sales_2002-2022_GL.xlsx", index=None, header=True)
```

Serial Number – An identifier for a transaction.

List Year – The year when the property was listed for assessment or sale.

Date Recorded – The date when the sale or assessment information was officially recorded.

Town – The name of the town or municipality where the property is located.

Address – The physical address of the property.

Assessed Value – The value assigned to the property by tax authorities for taxation purposes.

Sale Amount – The actual price at which the property was sold.

Sales Ratio – The ratio of the assessed value to the sale price (used for tax and appraisal analysis).

Property Type – The category of the property (e.g., residential, commercial, industrial, etc.).

Residential Type – If the property is residential, this specifies the type (e.g., single-family home, apartment, etc.).

Non Use Code – A code indicating if the property is not being used for its intended purpose (e.g., vacant land, government-owned).

Assessor Remarks – Comments or additional notes from the property assessor.

OPM Remarks – Remarks from the Office of Policy and Management (OPM), possibly related to tax policies or regulations.

Location – Geographic details or coordinates of the property.

Basic insights from the data

In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1097629 entries, 0 to 1097628
Data columns (total 14 columns):
 #   Column            Non-Null Count    Dtype  
---  ------            --------------    -----  
 0   Serial Number     1097629 non-null  int64  
 1   List Year         1097629 non-null  int64  
 2   Date Recorded     1097627 non-null  object 
 3   Town              1097629 non-null  object 
 4   Address           1097578 non-null  object 
 5   Assessed Value    1097629 non-null  float64
 6   Sale Amount       1097629 non-null  float64
 7   Sales Ratio       1097629 non-null  float64
 8   Property Type     715183 non-null   object 
 9   Residential Type  699240 non-null   object 
 10  Non Use Code      313451 non-null   object 
 11  Assessor Remarks  171228 non-null   object 
 12  OPM remarks       13031 non-null    object 
 13  Location          298111 non-null   object 
dtypes: float64(3), int64(2), object(9)
memory usage: 117.2+ MB
```

In [15]:
```
pd.DataFrame({
    'Count': df.count(),
    'Null': df.isnull().sum(),
    'Cardinality': df.nunique()
})
```

Out[15]:

	Count	Null	Cardinality
Serial Number	1097629	0	96220
List Year	1097629	0	22
Date Recorded	1097627	2	6958
Town	1097629	0	170
Address	1097578	51	771931
Assessed Value	1097629	0	99306
Sale Amount	1097629	0	61075
Sales Ratio	1097629	0	552974
Property Type	715183	382446	11
Residential Type	699240	398389	5
Non Use Code	313451	784178	105
Assessor Remarks	171228	926401	75286
OPM remarks	13031	1084598	6490
Location	298111	799518	216556

Only these columns do not have missing values :

- Serial Number / 1097629 non-null / int64
- List Year / 1097629 non-null / int64
- Town / 1097629 non-null / object
- Assessed Value / 1097629 non-null / float64
- Sale Amount / 1097629 non-null / float64
- Sales Ratio / 1097629 non-null / float64

These columns have few missing values :

- Date Recorded / 1097627 non-null object : **2 missing values**
- Address / 1097578 non-null / object : **51 missing values**

These columns have a significant amount of missing values :

- Property Type / 715183 non-null / object : **382446 missing values**
- Residential Type / 699240 non-null / object : **398389 missing values**
- Non Use Code / 313451 non-null / object : **313451 missing values**
- Assessor Remarks / 171228 non-null / object : **171228 missing values**
- OPM remarks / 13031 non-null / object : **13031 missing values**

- Location / 298111 non-null / object : **298111 missing values**

In [16]:
```
print('Number of duplicated rows :',df.duplicated().sum())
```

```
Number of duplicated rows : 0
```

There is no duplicated columns

In [17]:
```
non_missing_values_features =  ['Serial Number', 'List Year', 'Town', 'Assessed

few_missing_values_features = ['Date Recorded', 'Address']

significant_missing_values_features = ['Property Type', 'Residential Type', 'Non
```

How should we handle Missing data ? (Don't run, it's too long)

In [ ]:
```
df1 = df.copy()
```

Non Use Code, Assessor Remarks and OPM remarks

The columns 'Non Use Code', 'Assessor Remarks' and 'OPM remarks' contain a substantial amount of missing values and the study and model will not beneeding these features so it is better to drop these columns.

The column 'Serial number' is irrelevent and does not even differentiate each property. To differentiate between properties we use the 'Adderss' column as it is unique to each property

In [ ]:
```
df1.shape
```

Out[ ]:
```
(1097629, 14)
```

In [ ]:
```
df1.drop(columns=['Non Use Code','Assessor Remarks','OPM remarks'], axis=1, inpl
df1.head()
```

Out[ ]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	Sales Ratio	Prop
0	220008	2022	01/30/2023	Andover	618 ROUTE 6	139020.0	232000.0	0.5992	Reside
1	2020348	2020	09/13/2021	Ansonia	230 WAKELEE AVE	150500.0	325000.0	0.4630	Comme
2	20002	2020	10/02/2020	Ashford	390 TURNPIKE RD	253000.0	430000.0	0.5883	Reside
3	210317	2021	07/05/2022	Avon	53 COTSWOLD WAY	329730.0	805000.0	0.4096	Reside
4	200212	2020	03/09/2021	Avon	5 CHESTNUT DRIVE	130400.0	179900.0	0.7248	Reside

Property Type and Residential Type (Don't run because too long)

In [ ]:

```
print('The Residential types are :', df1['Residential Type'].dropna().unique())
```

```
The Residential types are : ['Single Family' 'Condo' 'Two Family' 'Four Family'
'Three Family']
```

In [ ]:

```
print('The Property types are :', df1['Property Type'].dropna().unique())
```

```
The Property types are : ['Residential' 'Commercial' 'Vacant Land' 'Apartments'
'Industrial'
 'Public Utility' 'Condo' 'Two Family' 'Three Family' 'Single Family'
 'Four Family']
```

The first thing to observe is that we have 2 types of properties : Residential and Non-Residential.

- **Residential properties :** Any property meant for people to live in.

It contains :

Single Family: A standalone home designed for one family.

Two Family: A building with two separate living units (also called a duplex).

Three Family: A building with 3 separate living units.

Four Family: A building with 4 separate living units.

Condo: A unit in a building or complex where you own your individual unit but share common areas.

- **Non Residential properties :**

It contains :

Commercial: Used for business activities (retail, offices, etc.).

Vacant Land: Land without any residential or commercial structure — often undeveloped.

Industrial: Used for manufacturing, storage, or distribution (factories, warehouses, etc.).

Apartments: Although people live there, it's often classified differently because:

They're usually investment properties or multi-unit rentals, not owned individually like condos or single-family homes.

Often treated differently for zoning, valuation, and taxation.

Public Utility: Land/buildings used for infrastructure (e.g., electrical substations, water plants, etc.).

First in the column 'Property type' we replace ['Single Family' 'Two Family' 'Condo' 'Four Family' 'Three Family'] by 'Residential' and put them in 'Residential Type'

In [ ]:
```
residential_mapping = {
    'Single Family': 'Single Family',
    'Two Family': 'Two Family',
    'Three Family': 'Three Family',
    'Four Family': 'Four Family',
    'Condo': 'Condo'
}

df1['Residential Type'] = df1['Property Type'].map(residential_mapping)

df1['Property Type'] = df1['Property Type'].replace(residential_mapping.keys(),
```

In [ ]:
```
pd.DataFrame({
    'Count': df1.count(),
    'Null': df1.isnull().sum(),
    'Cardinality': df1.nunique()
})
```

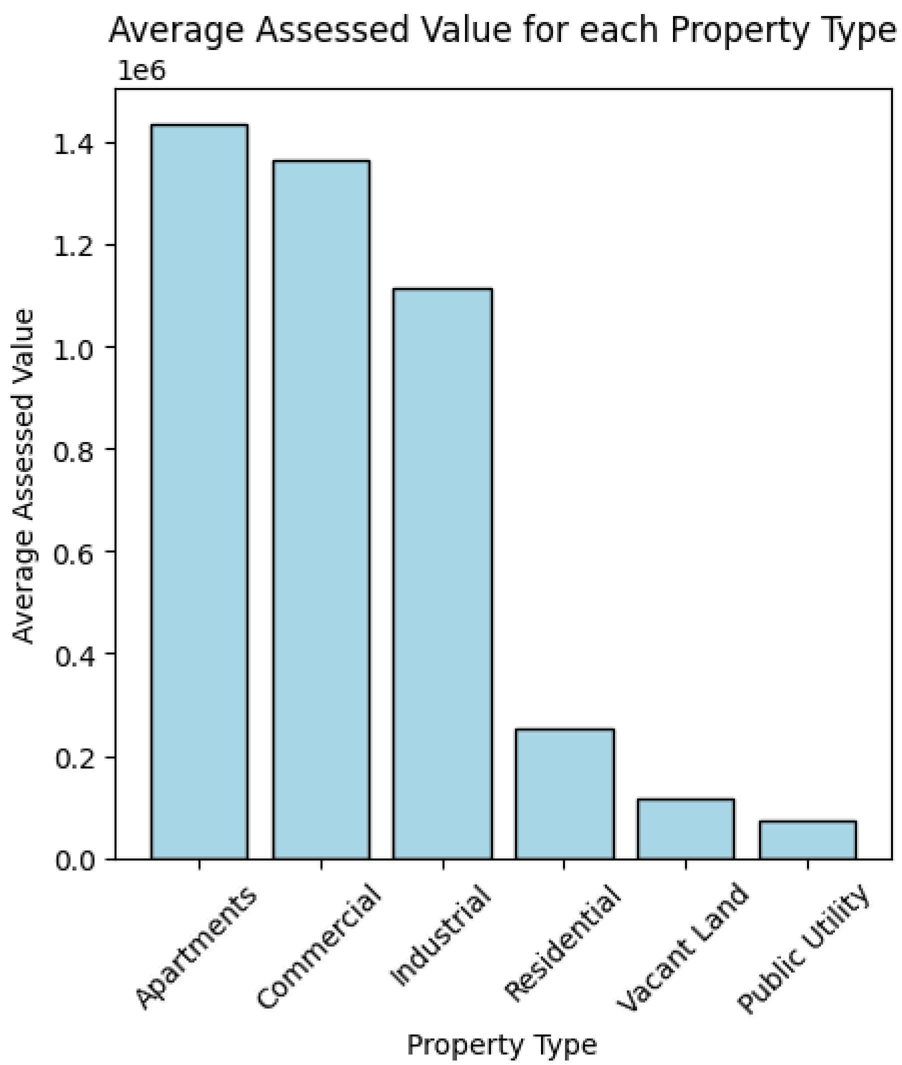
Out[ ]:

	Count	Null	Cardinality
Serial Number	1097629	0	96220
List Year	1097629	0	22
Date Recorded	1097627	2	6958
Town	1097629	0	170
Address	1097578	51	771931
Assessed Value	1097629	0	99306
Sale Amount	1097629	0	61075
Sales Ratio	1097629	0	552974
Property Type	715183	382446	6
Residential Type	548176	549453	5
Location	298111	799518	216556

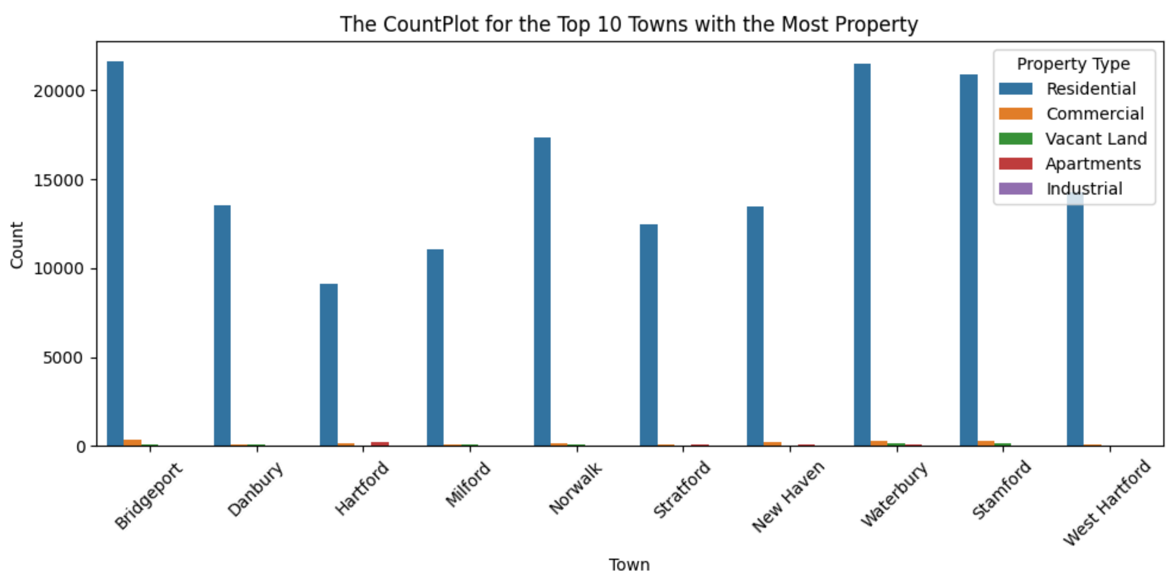
Let's first fill the missing values in the column 'Property Type'

In [ ]:

```
df_prop_ass = df1.groupby(['Property Type'])['Assessed Value'].mean().reset_inde
df_prop_ass = df_prop_ass.sort_values(by = 'Assessed Value', ascending=False)

fig = plt.figure(figsize=(5,5))

plt.bar(df_prop_ass['Property Type'],df_prop_ass['Assessed Value'], color = 'lig
plt.xlabel('Property Type')
plt.xticks(rotation=45)
plt.ylabel('Average Assessed Value')
plt.title('Average Assessed Value for each Property Type')

plt.show()
```




```
In [ ]: top10_towns = df['Town'].value_counts().head(10).index
fig = plt.figure(figsize=(10,5))
sns.countplot(x='Town', data=df1[df1['Town'].isin(top10_towns)], hue='Property T

plt.title('The CountPlot for the Top 10 Towns with the Most Property')
plt.xlabel('Town')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
```



In Property Type, the mean Assessed Value of the property types per town will be calculated. The Assessed Value will be compared to those mean values, and the closest mean value property type will be used to fill in the missing values.

In [ ]:
```
refrence1 = df1.groupby(['Property Type','Town'])['Assessed Value'].mean().reset

missing_df = df1[df1['Property Type'].isnull()].copy()
filled_df = df1[~df1['Property Type'].isnull()].copy()
```

In [ ]:
```
def find_closest_prop_type(town, value):
    candidates = refrence1[refrence1['Town'] == town]
    if candidates.empty:
        return np.nan
    closest = (candidates['Mean Assessed Value'] - value).abs().idxmin()
    return candidates.loc[closest, 'Property Type']
```

In [ ]:
```
missing_df['Property Type'] = missing_df.apply(
    lambda row: find_closest_prop_type(row['Town'], row['Assessed Value']),
    axis=1
)

df2 = pd.concat([filled_df, missing_df], ignore_index=True)
```

In [ ]:
```
print("The number of missing Property Type:", df2['Property Type'].isnull().sum(
```

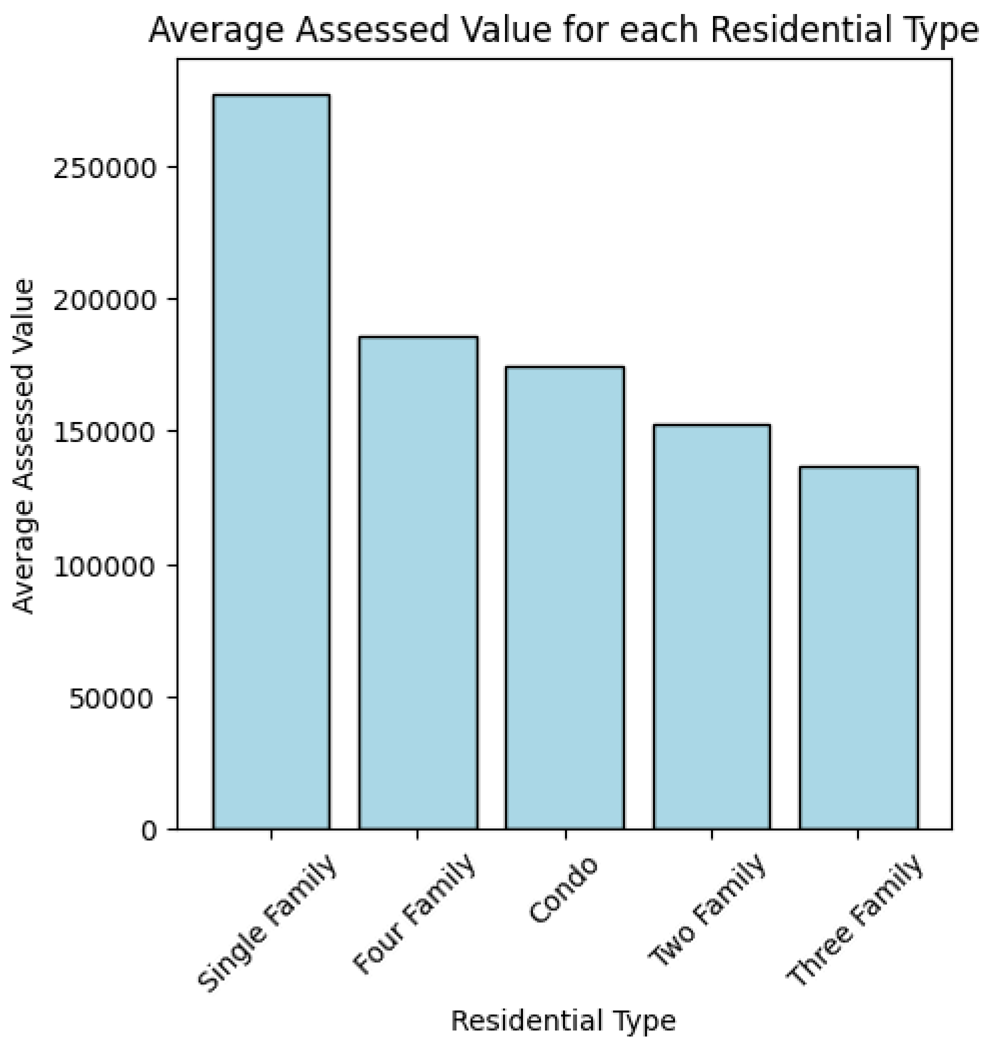
```
The number of missing Property Type: 0
```

Secondly, let's fill the missing values in the column 'Residential Type', but we know that we only need to fill the missing values where the property type is residential

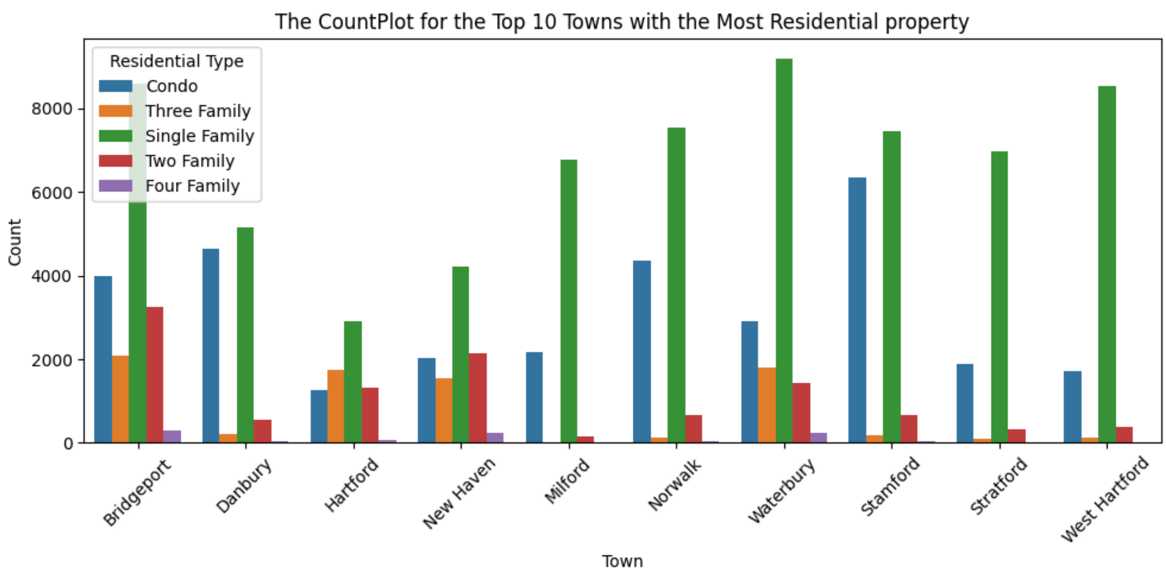
In [ ]:
```
df2.shape[0]
```

Out[ ]:
```
1097629
```

In [ ]:
```
df_prop_res = df2[df2['Property Type']=='Residential'].groupby(['Residential Typ
df_prop_res = df_prop_res.sort_values(by = 'Assessed Value', ascending=False)

fig = plt.figure(figsize=(5,5))

plt.bar(df_prop_res['Residential Type'],df_prop_res['Assessed Value'], color = '
plt.xlabel('Residential Type')
plt.xticks(rotation=45)
plt.ylabel('Average Assessed Value')
plt.title('Average Assessed Value for each Residential Type')

plt.show()
```



In [ ]:
```
top10_towns = df2['Town'].value_counts().head(10).index
fig = plt.figure(figsize=(10,5))
sns.countplot(x='Town', data=df2[df2['Town'].isin(top10_towns)], hue='Residentia

plt.title('The CountPlot for the Top 10 Towns with the Most Residential property
plt.xlabel('Town')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
```



```
In [ ]: refrence2 = df2.groupby(['Residential Type','Town'])['Assessed Value'].mean().re
        missing_df1 = df2[df2['Residential Type'].isnull()].copy()
        filled_df1 = df2[~df2['Residential Type'].isnull()].copy()
```

```
In [ ]: refrence2
```

Out[ ]:

	Residential Type	Town	Mean Assessed Value
0	Condo	Ansonia	109083.590361
1	Condo	Ashford	41580.000000
2	Condo	Avon	191328.011745
3	Condo	Beacon Falls	147419.314381
4	Condo	Berlin	414452.030464
...	...	...	...
711	Two Family	Windsor Locks	128755.064935
712	Two Family	Wolcott	179531.600000
713	Two Family	Woodbridge	168742.258065
714	Two Family	Woodbury	262642.307692
715	Two Family	Woodstock	191787.500000

716 rows × 3 columns

```
In [ ]: missing_df1.shape[0]+filled_df1.shape[0]

Out[ ]: 1097629
```

```
In [ ]: def find_closest_res_type(town, value):
            candidates = refrence2[refrence2['Town'] == town]
            if candidates.empty:
                return np.nan
            closest = (candidates['Mean Assessed Value'] - value).abs().idxmin()
            return candidates.loc[closest, 'Residential Type']
```

```
In [ ]: missing_df1['Residential Type'] = missing_df1.apply(
            lambda row: find_closest_res_type(row['Town'], row['Assessed Value']),
            axis=1
        )
```

```
In [ ]: df3 = pd.concat([filled_df1, missing_df1], ignore_index=True)
```

```
In [ ]: df3.loc[df3['Property Type']!='Residential','Residential Type']='Non Residential
```

```
In [ ]: print("The number of missing Residential Type:", df3['Residential Type'].isnull(

        The number of missing Residential Type: 0
```

```
In [ ]: print("The number of non residential properties:",df3[df3['Property Type']!='Res

        The number of non residential properties: 264027
```

```
In [ ]: print("The number of non residential properties:",df3[df3['Residential Type']=='
```

The number of non residential properties: 264027

```
In [ ]: print("The number of  residential properties:",df3[df3['Property Type']=='Reside
```

The number of residential properties: 833602

```
In [ ]: df3.shape[0]
```

Out[ ]: 1097629

```
In [ ]: df3.to_csv('data.csv')
```

Date Recorded and Address

```
In [ ]: filepath = '/content/drive/MyDrive/Colab Notebooks/Projects/Personal/Real estate
```

```
In [ ]: data = pd.read_csv(filepath)
```

```
In [ ]: data = data.drop(columns='Unnamed: 0')
```

```
In [ ]: data.head()
```

Out[ ]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	Sales Ratio
0	60228	2006	07/05/2007	Bethel	10 HUNTINGTON COURT	120960.0	250000.0	0.483840
1	60075	2006	04/05/2007	Essex	7 PRATT ST	143400.0	339500.0	0.422386
2	60416	2006	05/25/2007	Newington	29 STERLING DR	221970.0	340000.0	0.652853
3	60537	2006	08/31/2007	Branford	91 JEFFERSON WOODS	118800.0	210000.0	0.565714
4	60421	2006	05/08/2007	Glastonbury	9 BOXWOOD LN	84000.0	174000.0	0.482759

```
In [ ]: data.shape
```

Out[ ]: (1097629, 11)

```
In [ ]: data['Date Recorded'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 1097629 entries, 0 to 1097628
Series name: Date Recorded
Non-Null Count    Dtype
--------------    -----
1097627 non-null  object
dtypes: object(1)
memory usage: 8.4+ MB
```

In [ ]:
```
print(data['Date Recorded'].isnull().sum())
```

2

We only have 2 missing values

In [ ]:
```
data[data['Date Recorded'].isnull()]
```

Out[ ]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	Sales Ratio	Propert Typ
826133	20280	2002	NaN	Orange	NaN	0.0	0.0	0.0	Vacai Lar
827626	0	2002	NaN	Orange	NaN	0.0	0.0	0.0	Vacai Lar

In [ ]:
```
data[data['Address'].isnull()]
```

Out[ ]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	S R
11699	60474	2006	07/30/2007	Farmington	NaN	0.0	453369.0	0.000
423711	170165	2017	12/08/2017	Manchester	NaN	129300.0	224000.0	0.577
450631	172767	2017	01/12/2018	Shelton	NaN	227500.0	500000.0	0.455
454132	17001	2017	10/02/2017	North Haven	NaN	193130.0	242000.0	0.798
715440	39999	2003	02/02/2004	West Haven	NaN	0.0	0.0	0.000
715476	49996	2004	05/17/2005	Lisbon	NaN	0.0	0.0	0.000
715502	48886	2004	06/13/2005	Lisbon	NaN	0.0	0.0	0.000
737239	10537	2001	02/05/2002	Hartford	NaN	0.0	120000.0	0.000
740908	10640	2001	12/19/2001	Bridgeport	NaN	2106020.0	45000.0	46.800
826133	20280	2002	NaN	Orange	NaN	0.0	0.0	0.000
827626	0	2002	NaN	Orange	NaN	0.0	0.0	0.000
854278	30125	2003	11/10/2003	New Milford	NaN	55090.0	400000.0	0.137
867388	39998	2003	08/12/2004	Lisbon	NaN	0.0	0.0	0.000
876058	30100	2003	05/20/2004	North Stonington	NaN	7210.0	149000.0	0.048
880656	39995	2003	02/02/2004	West Haven	NaN	0.0	0.0	0.000
887426	39998	2003	02/20/2004	West Haven	NaN	0.0	0.0	0.000
888018	40088	2004	11/01/2004	Groton	NaN	0.0	7060035.0	0.000
888554	48889	2004	06/02/2005	Lisbon	NaN	0.0	0.0	0.000
901755	40080	2004	11/24/2004	Brookfield	NaN	147340.0	1015000.0	0.145
904646	39997	2003	02/02/2004	West Haven	NaN	0.0	0.0	0.000

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	S R
906620	48888	2004	03/03/2005	Bridgeport	NaN	0.0	0.0	0.000
910606	49999	2004	05/12/2005	Lisbon	NaN	0.0	0.0	0.000
920674	48887	2004	06/30/2005	Lisbon	NaN	0.0	0.0	0.000
921289	40070	2004	10/12/2004	Hartford	NaN	54950.0	105000.0	0.523
923244	40059	2004	04/29/2005	Harwinton	NaN	0.0	450000.0	0.000
926163	48884	2004	05/13/2005	Lisbon	NaN	0.0	0.0	0.000
930216	48885	2004	05/13/2005	Lisbon	NaN	0.0	0.0	0.000
931826	49997	2004	05/27/2005	Lisbon	NaN	0.0	0.0	0.000
933017	49994	2004	05/12/2005	Lisbon	NaN	0.0	0.0	0.000
933829	48811	2004	06/22/2005	Lisbon	NaN	0.0	0.0	0.000
935529	48810	2004	06/02/2005	Lisbon	NaN	0.0	0.0	0.000
936017	49993	2004	05/23/2005	Lisbon	NaN	0.0	0.0	0.000
936089	40318	2004	06/07/2005	East Hampton	NaN	0.0	400000.0	0.000
936751	48888	2004	06/08/2005	Lisbon	NaN	0.0	0.0	0.000
937135	49998	2004	05/12/2005	Lisbon	NaN	0.0	0.0	0.000
937558	41588	2004	09/20/2005	Meriden	NaN	0.0	100.0	0.000
939956	49995	2004	05/12/2005	Lisbon	NaN	0.0	0.0	0.000
957098	49999	2004	07/05/2005	Salem	NaN	0.0	0.0	0.000
959194	49998	2004	07/05/2005	Salem	NaN	0.0	0.0	0.000
959744	41230	2004	03/10/2005	Waterbury	NaN	234500.0	425000.0	0.551

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	S R
961867	48992	2004	05/12/2005	Salem	NaN	0.0	0.0	0.000
963336	40198	2004	06/02/2005	Norfolk	NaN	50200.0	350000.0	0.143
965708	49997	2004	07/08/2005	Salem	NaN	0.0	0.0	0.000
982389	40285	2004	01/11/2005	Torrington	NaN	0.0	155800.0	0.000
988123	50862	2005	05/16/2006	Bristol	NaN	104100.0	155000.0	0.671
1044589	60058	2006	09/17/2007	Lyme	NaN	0.0	3656.0	0.000
1047174	60054	2006	12/08/2006	New Fairfield	NaN	0.0	3500.0	0.000
1047194	60032	2006	02/21/2007	Sterling	NaN	181310.0	301500.0	0.601
1047200	60159	2006	08/21/2007	Litchfield	NaN	55190.0	60000.0	0.919
1048112	60043	2006	07/19/2007	Pomfret	NaN	445340.0	875000.0	0.508
1050292	60237	2006	04/02/2007	South Windsor	NaN	72340.0	225000.0	0.321

We can see that the properties that don't include the address are not interesting in our repeated sales method since Address is ou identifier of a unique property. So we should drop the rows with the missing values in 'Address' Column.

The 2 missing values of Date Recorded are on that category.

In [ ]:
```
data2 = data[~data['Address'].isnull()].copy()
```

In [ ]:
```
data2.shape
```

Out[ ]:
```
(1097578, 11)
```

In [ ]:
```
data2['Date Recorded'] = pd.to_datetime(data2['Date Recorded'])
```

In [ ]:
```
data2['Date Recorded'].info()
```

```
<class 'pandas.core.series.Series'>
Index: 1097578 entries, 0 to 1097628
Series name: Date Recorded
Non-Null Count    Dtype
--------------    -----
1097578 non-null  datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 16.7 MB
```

Location

In [ ]:
```
pd.DataFrame({
    'Count': data2.count(),
    'Null': data2.isnull().sum(),
    'Cardinality': data2.nunique()
})
```

Out[ ]:

	Count	Null	Cardinality
Serial Number	1097578	0	96217
List Year	1097578	0	22
Date Recorded	1097578	0	6958
Town	1097578	0	170
Address	1097578	0	771931
Assessed Value	1097578	0	99306
Sale Amount	1097578	0	61072
Sales Ratio	1097578	0	552966
Property Type	1097578	0	6
Residential Type	1097578	0	6
Location	298106	799472	216554

In [ ]:
```
data2.head()
```

Out[ ]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	Sales Ratio
0	60228	2006	2007-07-05	Bethel	10 HUNTINGTON COURT	120960.0	250000.0	0.483840
1	60075	2006	2007-04-05	Essex	7 PRATT ST	143400.0	339500.0	0.422386
2	60416	2006	2007-05-25	Newington	29 STERLING DR	221970.0	340000.0	0.652853
3	60537	2006	2007-08-31	Branford	91 JEFFERSON WOODS	118800.0	210000.0	0.565714
4	60421	2006	2007-05-08	Glastonbury	9 BOXWOOD LN	84000.0	174000.0	0.482759

In [ ]:

```
data2['Town'].unique()
```

Out[ ]:

```
array(['Bethel', 'Essex', 'Newington', 'Branford', 'Glastonbury',
       'Ledyard', 'Danbury', 'Marlborough', 'Cromwell', 'Bristol',
       'Fairfield', 'Norwalk', 'Woodbury', 'Simsbury', 'Wallingford',
       'Watertown', 'Norwich', 'Stonington', 'Avon', 'Canton', 'Meriden',
       'Milford', 'New Haven', 'Sharon', 'Darien', 'Derby', 'Rocky Hill',
       'Greenwich', 'Enfield', 'Thompson', 'Groton', 'Westport', 'Vernon',
       'Windsor', 'East Haven', 'Trumbull', 'Southington', 'Clinton',
       'South Windsor', 'Suffield', 'Shelton', 'Farmington', 'Hartford',
       'Hamden', 'Southbury', 'Granby', 'Bridgeport', 'Monroe',
       'Guilford', 'Litchfield', 'Winchester', 'Waterbury', 'Woodstock',
       'Stratford', 'Berlin', 'Ellington', 'Bloomfield', 'Colchester',
       'New London', 'East Lyme', 'Somers', 'Wethersfield', 'Salem',
       'Manchester', 'Putnam', 'New Canaan', 'Wilton', 'Stamford',
       'Madison', 'Thomaston', 'Torrington', 'Plainville', 'Killingly',
       'Seymour', 'Stafford', 'Brookfield', 'New Milford', 'West Haven',
       'Ansonia', 'Tolland', 'North Haven', 'Griswold', 'Prospect',
       'Ridgefield', 'New Britain', 'Middletown', 'West Hartford',
       'East Haddam', 'Bethany', 'Burlington', 'Lyme', 'East Granby',
       'East Windsor', 'Naugatuck', 'Haddam', 'Cheshire', 'Ashford',
       'Mansfield', 'Harwinton', 'Newtown', 'Deep River', 'East Hampton',
       'Coventry', 'Chester', 'Durham', 'Roxbury', 'Canterbury',
       'North Canaan', 'New Fairfield', 'Franklin', 'Bozrah', 'Brooklyn',
       'New Hartford', 'Bethlehem', 'Goshen', 'Lebanon', 'Morris',
       'Weston', 'Barkhamsted', 'North Branford', 'Hartland', 'Eastford',
       'Kent', 'Colebrook', 'Salisbury', 'Plainfield', 'Bolton',
       'Norfolk', 'Hampton', 'Chaplin', 'Preston', 'Canaan', 'Windham',
       'Sherman', 'Waterford', 'Windsor Locks', 'Redding', 'Old Lyme',
       'Old Saybrook', 'Woodbridge', 'North Stonington', 'Union',
       'Warren', 'Voluntown', 'Washington', 'Oxford', 'Willington',
       'Sterling', 'Scotland', 'Pomfret', 'Sprague', 'Portland',
       'Montville', 'East Hartford', 'Columbia', 'Middlebury',
       'Bridgewater', 'Cornwall', 'Beacon Falls', 'Lisbon',
       'Killingworth', 'Plymouth', 'Orange', 'Easton', 'Andover',
       'Middlefield', 'Hebron', 'Wolcott', '***Unknown***', 'Westbrook'],
      dtype=object)
```

In [ ]:
```
data2[data2['Town']=="***Unknown***"]
```

Out[ ]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	Sale Rati
42710	70086	2007	2007-12-18	***Unknown***	18 MATHIEU LANE	66540.0	282450.0	0.23558

In [ ]:
```
data2[data2['Address']=='18 MATHIEU LANE']
```

Out[ ]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	Sa Ra
42710	70086	2007	2007-12-18	***Unknown***	18 MATHIEU LANE	66540.0	282450.0	0.235
42880	70086	2007	2007-12-18	East Hampton	18 MATHIEU LANE	66540.0	282450.0	0.235
1054061	70085	2007	2007-12-18	East Hampton	18 MATHIEU LANE	66540.0	50000.0	1.330

In [ ]:
```
data2 = data2[data2['Town']!='***Unknown***']
```

In [ ]:
```
#!pip install geopy
```

```
Requirement already satisfied: geopy in /usr/local/lib/python3.11/dist-packages
(2.4.1)
Requirement already satisfied: geographiclib<3,>=1.52 in /usr/local/lib/python3.1
1/dist-packages (from geopy) (2.0)
```

In [ ]:
```
from geopy.geocoders import Nominatim

loc = Nominatim(user_agent="Geopy Library")

getLoc = loc.geocode("LOT 2 DINGS RD New Hartford, CT, USA")

print(getLoc.address)

print("Latitude = ", getLoc.latitude, "\n")
print("Longitude = ", getLoc.longitude)
```

```
Dings Road, Bakerville, New Hartford, Northwest Hills Planning Region, Connecticu
t, 06057, United States
Latitude =  41.836973044198125

Longitude =  -73.01522052995116
```

In [ ]:
```
data2
```

Out[ ]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount
0	60228	2006	2007-07-05	Bethel	10 HUNTINGTON COURT	120960.0	250000.0
1	60075	2006	2007-04-05	Essex	7 PRATT ST	143400.0	339500.0
2	60416	2006	2007-05-25	Newington	29 STERLING DR	221970.0	340000.0
3	60537	2006	2007-08-31	Branford	91 JEFFERSON WOODS	118800.0	210000.0
4	60421	2006	2007-05-08	Glastonbury	9 BOXWOOD LN	84000.0	174000.0
...	...	...	...	...	...	...	...
1097624	19150	2019	2020-01-13	Newtown	22 WASHINGTON AVENUE	53640.0	122500.0
1097625	190242	2019	2020-09-18	Weston	OLD HYDE ROAD	181440.0	150000.0
1097626	19000067	2019	2020-05-19	New Hartford	LOT 2 DINGS RD	87955.0	35000.0
1097627	190713	2019	2020-06-01	New Haven	1083 WHALLEY AV	262220.0	325000.0
1097628	190344	2019	2019-12-20	Milford	250 RESEARCH DR	4035970.0	7450000.0

1097577 rows × 11 columns

In [ ]:

```
address = data2.loc[0,"Address"] +" "+data2.loc[0,'Town']
f"{address}, CT, USA"
```

Out[ ]: '10 HUNTINGTON COURT Bethel, CT, USA'

In [ ]:

```
data3 = data2.copy()
```

In [ ]:

```
#from geopy.geocoders import Nominatim
#from geopy.exc import GeocoderTimedOut, GeocoderUnavailable
#import numpy as np
#import pandas as pd
#import time

#geolocator = Nominatim(user_agent="your_app_name")


#I = []
```

```
#for i in range(len(data3)):
  # Use .loc for label-based indexing to access 'Location' column for each row
#  if pd.notna(data3.loc[i,'Location']):
#    point_str = data3.loc[i,'Location']
#    coords = point_str.replace("POINT (", "").replace(")", "")
#    longitude, latitude = map(float, coords.split())
#    data3.loc[i, "Latitude"] = latitude
    data3.loc[i, "Longitude"] = longitude
#  else :
#      try:
#          address = data2.loc[i, "Address"] + " " + data2.loc[i, 'Town']
#          location = geolocator.geocode(f"{address}, CT, USA", timeout=10)
#
#          if location is not None:
#              data3.loc[i, "Latitude"] = location.latitude
#              data3.loc[i, "Longitude"] = location.longitude
#          else:
#              I.append(i)
#              print(f"Address not found in {i}: {address}")
#
#      except (GeocoderTimedOut, GeocoderUnavailable) as e:
#          print(f"Geocoding error for index {i}: {e}")
```

```
Address not found in 3: 91 JEFFERSON WOODS Branford
Address not found in 5: 120 GALLUP HL RD 1D Ledyard
Address not found in 6: 163 SOUTH ST UT 1 Danbury
Address not found in 11: 279 REDSTONE HL RD UT63B Bristol
Address not found in 14: 7 UPPER CMNS Woodbury
Address not found in 22: 245 CHERRY AVE UT C11 Watertown
Address not found in 24: 2 FROST ST 4 Norwalk
Address not found in 25: 1 WEST ST UT 118 Simsbury
Address not found in 26: 97 HILLTOP DR Simsbury
Address not found in 30: 14 BAYPATH WAY Branford
Address not found in 34: 61 LA MIRAGE Meriden
Address not found in 36: 32 ABERDEEN RD Fairfield
Address not found in 38: 8 HUGHES PL #2E New Haven
Address not found in 39: 4 UPPER MAIN ST #1 Sharon
Address not found in 41: 121 ORANGEWOOD EAST Derby
Address not found in 46: 2202 HARBOUR VIEW DR Rocky Hill
Address not found in 49: 52 LAFAYETTE PL #1I Greenwich
Address not found in 50: 142 MAIN ST #4 Norwalk
```

This seems a good idea but it takes too mach time and not all addresses are recognizable

Maybe we should drop the 'Location' since it is not used in our model.

In [ ]:
```
data3.drop(columns='Location',inplace=True)
```

In [ ]:
```
data3.head()
```

Out[ ]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	Sales Ratio
0	60228	2006	2007-07-05	Bethel	10 HUNTINGTON COURT	120960.0	250000.0	0.483840
1	60075	2006	2007-04-05	Essex	7 PRATT ST	143400.0	339500.0	0.422386
2	60416	2006	2007-05-25	Newington	29 STERLING DR	221970.0	340000.0	0.652853
3	60537	2006	2007-08-31	Branford	91 JEFFERSON WOODS	118800.0	210000.0	0.565714
4	60421	2006	2007-05-08	Glastonbury	9 BOXWOOD LN	84000.0	174000.0	0.482759

In [ ]:

```
data3.shape
```

Out[ ]:

```
(1097577, 10)
```

Outlier Detection(Don't run, it's too long)

In [ ]:

```
#data3.to_csv('cleaned_data.csv')
```

In [ ]:

```
filepath = '/content/drive/MyDrive/Colab Notebooks/Projects/Personal/Real estate
```

In [ ]:

```
data4 = pd.read_csv(filepath)
data4.drop(columns='Unnamed: 0',inplace=True)
```

In [ ]:

```
data4.head()
```

Out[ ]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	Sales Ratio
0	60228	2006	2007-07-05	Bethel	10 HUNTINGTON COURT	120960.0	250000.0	0.483840
1	60075	2006	2007-04-05	Essex	7 PRATT ST	143400.0	339500.0	0.422386
2	60416	2006	2007-05-25	Newington	29 STERLING DR	221970.0	340000.0	0.652853
3	60537	2006	2007-08-31	Branford	91 JEFFERSON WOODS	118800.0	210000.0	0.565714
4	60421	2006	2007-05-08	Glastonbury	9 BOXWOOD LN	84000.0	174000.0	0.482759

In [ ]:

```
data4.shape
```

Out[ ]:

```
(1097577, 10)
```

Are there outliers in Sale Amount or Assessed Value, and how should they be handled?

The extreme values of Assessed Value and Sale Amount are rejected on the basis of the Box Plot method, to reduce their impact.

Data are considered outliers if their value is outside of this interval:

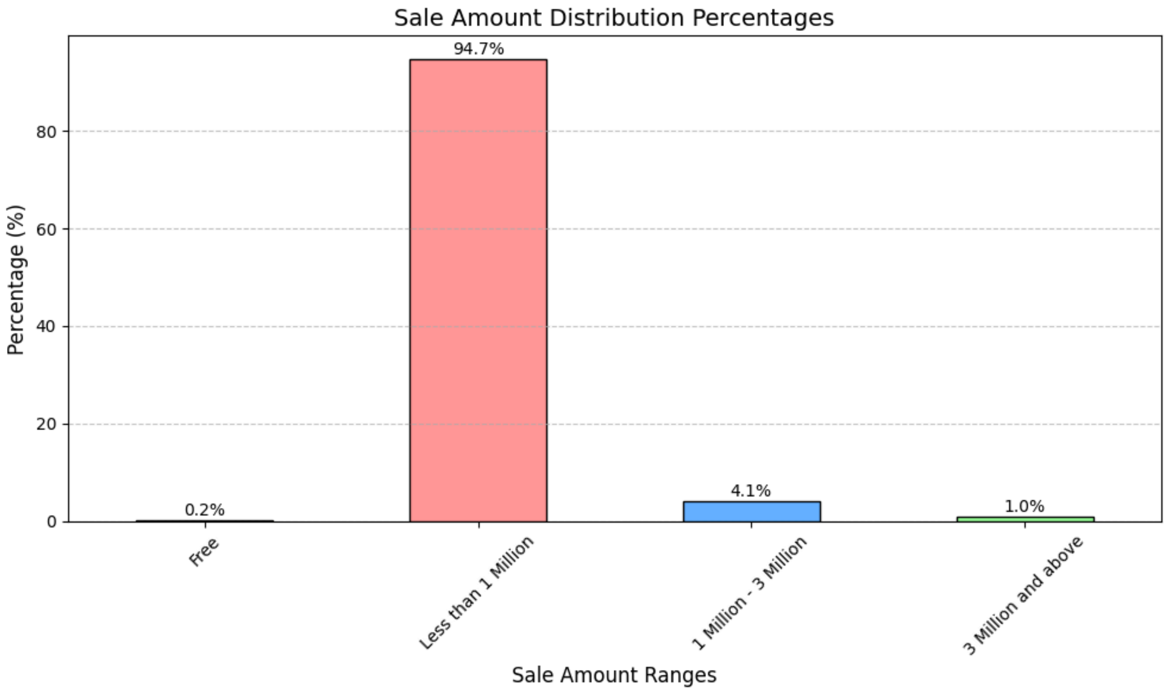
[Assessed Value ± 1.5 * (Q3-Q1)] with **Qi is the ith quartile Assessed Value.**

[Sale Amount ± 1.5 * (Q3-Q1)] with **Qi is the ith quartile Sale Amount.**

In [ ]:

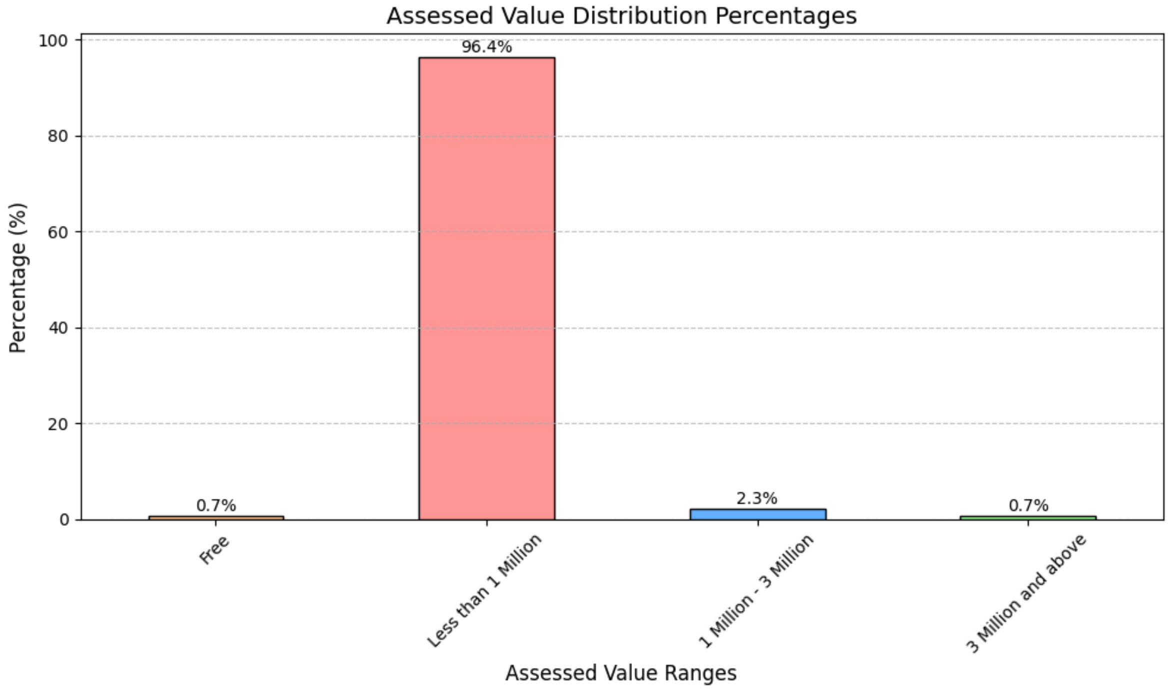
```
bins = [0, 1, 1000000, 3000000, float('inf')]
labels = ['Free', 'Less than 1 Million', '1 Million - 3 Million', '3 Million and

category_counts = pd.cut(data4['Sale Amount'], bins=bins, labels=labels, right=F

plt.figure(figsize=(10, 6))
colors = ['#FFCC99', '#FF9999', '#66B3FF', '#99FF99']
category_counts.sort_index().plot(kind='bar', color=colors, edgecolor='black')

plt.title('Sale Amount Distribution Percentages', fontsize=14)
plt.xlabel('Sale Amount Ranges', fontsize=12)
plt.ylabel('Percentage (%)', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)

for i, value in enumerate(category_counts.sort_index()):
    plt.text(i, value + 1, f'{value:.1f}%', ha='center', fontsize=10)
```

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

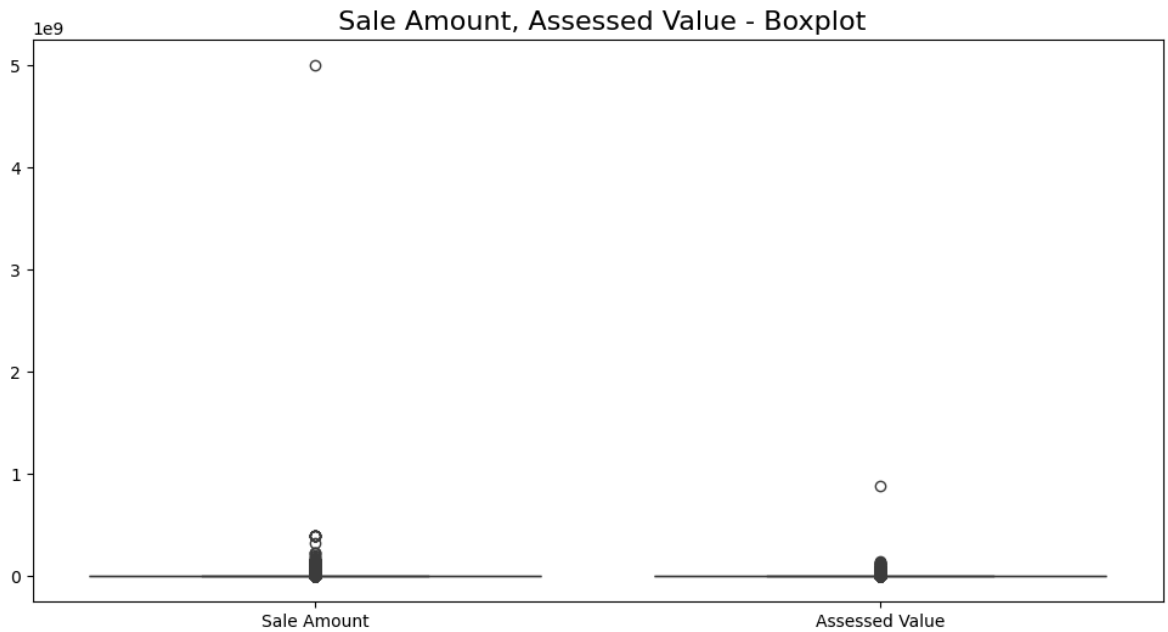


In [ ]:

```
bins = [0, 1, 1000000, 3000000, float('inf')]
labels = ['Free', 'Less than 1 Million', '1 Million - 3 Million', '3 Million and

category_counts = pd.cut(data4['Assessed Value'], bins=bins, labels=labels, righ

plt.figure(figsize=(10, 6))
colors = ['#FFCC99', '#FF9999', '#66B3FF', '#99FF99']
category_counts.sort_index().plot(kind='bar', color=colors, edgecolor='black')

plt.title('Assessed Value Distribution Percentages', fontsize=14)
plt.xlabel('Assessed Value Ranges', fontsize=12)
plt.ylabel('Percentage (%)', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)

for i, value in enumerate(category_counts.sort_index()):
    plt.text(i, value + 1, f'{value:.1f}%', ha='center', fontsize=10)

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



In [ ]:
```
plt.figure(figsize=(12, 6))

sns.boxplot(data=data4[['Sale Amount', 'Assessed Value']])

plt.title(" Sale Amount, Assessed Value - Boxplot", fontsize=16)

plt.show()
```

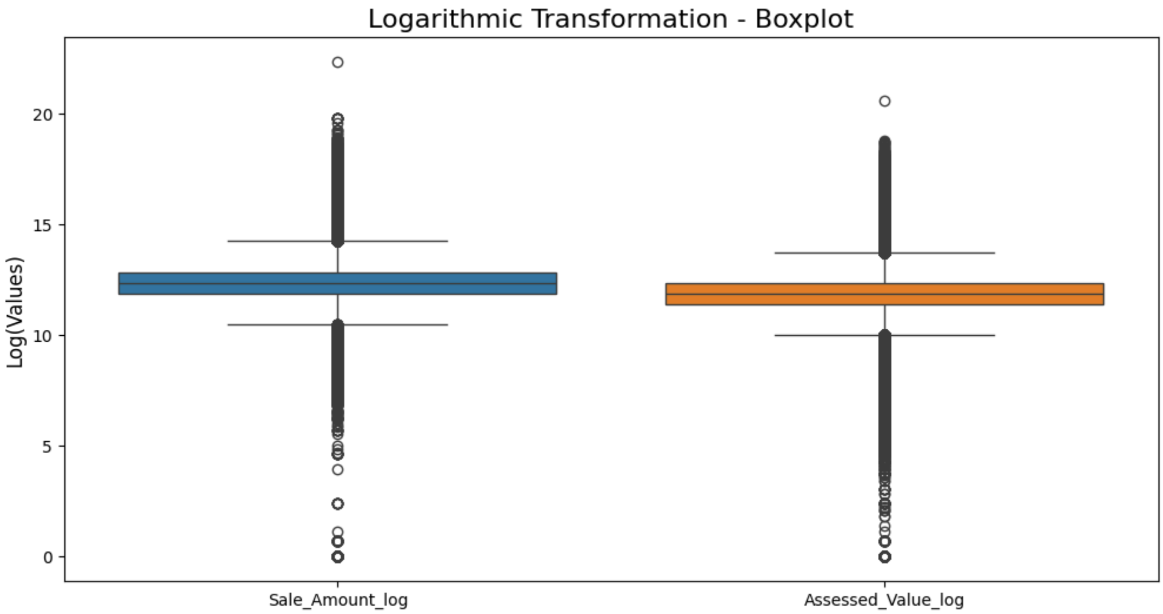


In [ ]:
```
print(data4[['Sale Amount', 'Assessed Value']].describe())
```

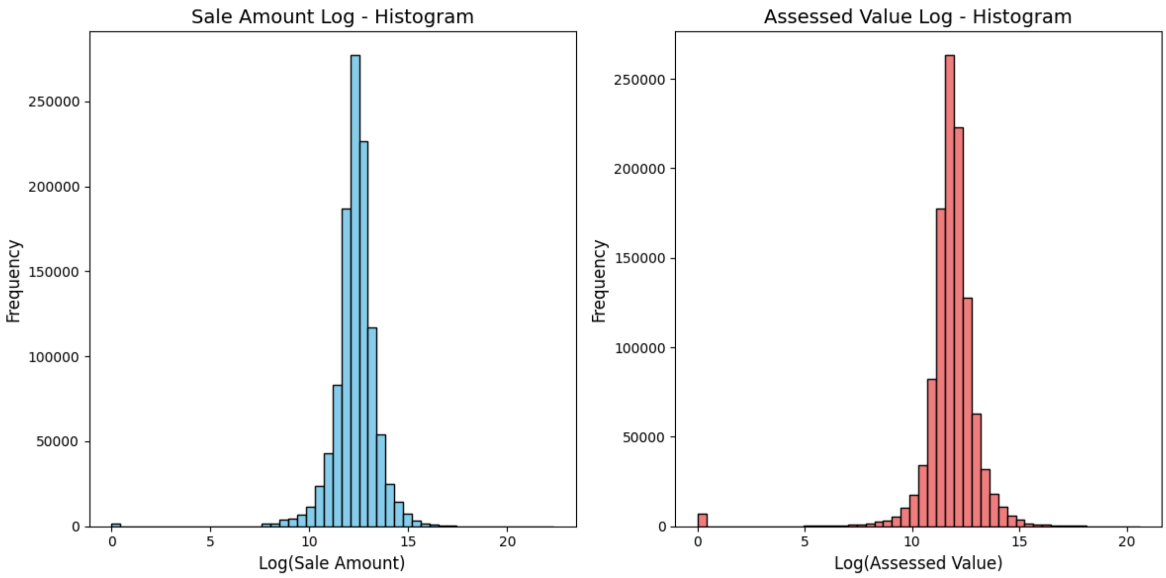
```
         Sale Amount  Assessed Value
count  1.097577e+06    1.097577e+06
mean   4.053210e+05    2.818112e+05
std    5.143610e+06    1.657928e+06
min    0.000000e+00    0.000000e+00
25%    1.450000e+05    8.910000e+04
50%    2.330000e+05    1.405900e+05
75%    3.750000e+05    2.282700e+05
max    5.000000e+09    8.815100e+08
```

In [ ]:
```python
data4['Sale_Amount_log'] = np.log1p(data4['Sale Amount'])
data4['Assessed_Value_log'] = np.log1p(data4['Assessed Value'])
```

In [ ]:
```python
plt.figure(figsize=(12, 6))

sns.boxplot(data=data4[['Sale_Amount_log', 'Assessed_Value_log']])

plt.title("Logarithmic Transformation - Boxplot", fontsize=16)
plt.ylabel("Log(Values)", fontsize=12)
plt.show()
```



In [ ]:
```python
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(data4['Sale_Amount_log'], bins=50, color='skyblue', edgecolor='black')
plt.title("Sale Amount Log - Histogram", fontsize=14)
plt.xlabel("Log(Sale Amount)", fontsize=12)
plt.ylabel("Frequency", fontsize=12)

plt.subplot(1, 2, 2)
plt.hist(data4['Assessed_Value_log'], bins=50, color='lightcoral', edgecolor='bl
plt.title("Assessed Value Log - Histogram", fontsize=14)
plt.xlabel("Log(Assessed Value)", fontsize=12)
plt.ylabel("Frequency", fontsize=12)

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



In [ ]:

```
log_summary = data4[['Sale_Amount_log', 'Assessed_Value_log']].describe()
print(log_summary)
```

```
       Sale_Amount_log  Assessed_Value_log
count     1.097577e+06        1.097577e+06
mean      1.231591e+01        1.180245e+01
std       1.093597e+00        1.363672e+00
min       0.000000e+00        0.000000e+00
25%       1.188450e+01        1.139753e+01
50%       1.235880e+01        1.185361e+01
75%       1.283468e+01        1.233829e+01
max       2.233270e+01        2.059715e+01
```

We should take out the outliers

Since our model is going to fit a linear regression on the logarithmic Sale amount(or Assessed value) as the response variable. We will also take out the ouliers on these variables.

In [ ]:

```
Q1 = data4[['Sale_Amount_log', 'Assessed_Value_log']].quantile(0.25)
Q3 = data4[['Sale_Amount_log', 'Assessed_Value_log']].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

lower_bound = lower_bound.reindex(data4[['Sale_Amount_log', 'Assessed_Value_log'
upper_bound = upper_bound.reindex(data4[['Sale_Amount_log', 'Assessed_Value_log'

outliers = (data4[['Sale_Amount_log', 'Assessed_Value_log']] < lower_bound) | (d


print("Number of Outliers - Sale Amount:", outliers['Sale_Amount_log'].sum())
print("Number of Outliers - Assessed Value:", outliers['Assessed_Value_log'].sum
```

```
Number of Outliers - Sale Amount: 69115
Number of Outliers - Assessed Value: 73166
```

In [ ]:

```
data5 = data4[~outliers.any(axis=1)].copy()
```

In [ ]:

```
data5
```

Out[ ]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	
0	60228	2006	2007-07-05	Bethel	10 HUNTINGTON COURT	120960.0	250000.0	0.4
1	60075	2006	2007-04-05	Essex	7 PRATT ST	143400.0	339500.0	0.4
2	60416	2006	2007-05-25	Newington	29 STERLING DR	221970.0	340000.0	0.6
3	60537	2006	2007-08-31	Branford	91 JEFFERSON WOODS	118800.0	210000.0	0.5
4	60421	2006	2007-05-08	Glastonbury	9 BOXWOOD LN	84000.0	174000.0	0.4
...	...	...	...	...	...	...	...	
1097571	190234	2019	2020-07-20	Wilton	481 DANBURY RD	445200.0	410000.0	1.0
1097572	19150	2019	2020-01-13	Newtown	22 WASHINGTON AVENUE	53640.0	122500.0	0.4
1097573	190242	2019	2020-09-18	Weston	OLD HYDE ROAD	181440.0	150000.0	1.2
1097574	19000067	2019	2020-05-19	New Hartford	LOT 2 DINGS RD	87955.0	35000.0	2.5
1097575	190713	2019	2020-06-01	New Haven	1083 WHALLEY AV	262220.0	325000.0	0.8

995169 rows × 12 columns

In [ ]:

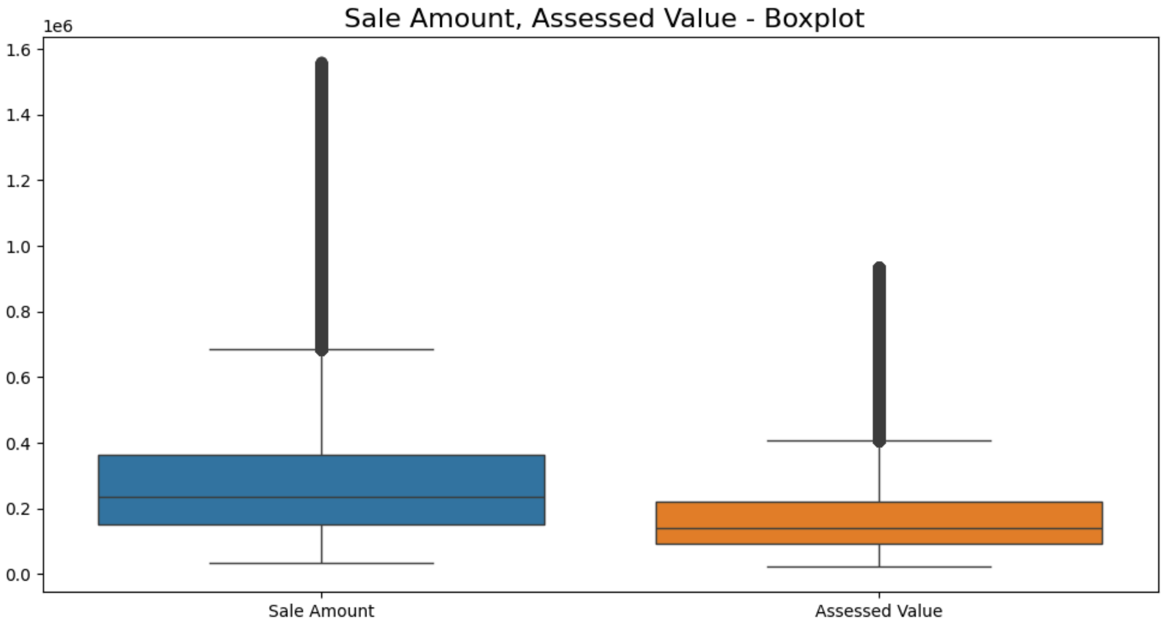
```
print(data5[['Sale Amount', 'Assessed Value']].describe())
```

```
        Sale Amount  Assessed Value
count  9.951690e+05    995169.000000
mean   2.962926e+05    180995.269787
std    2.244255e+05    136976.331376
min    3.489400e+04     21730.000000
25%    1.520000e+05     94010.000000
50%    2.350000e+05    141610.000000
75%    3.650000e+05    219100.000000
max    1.559320e+06    936000.000000
```

In [ ]:

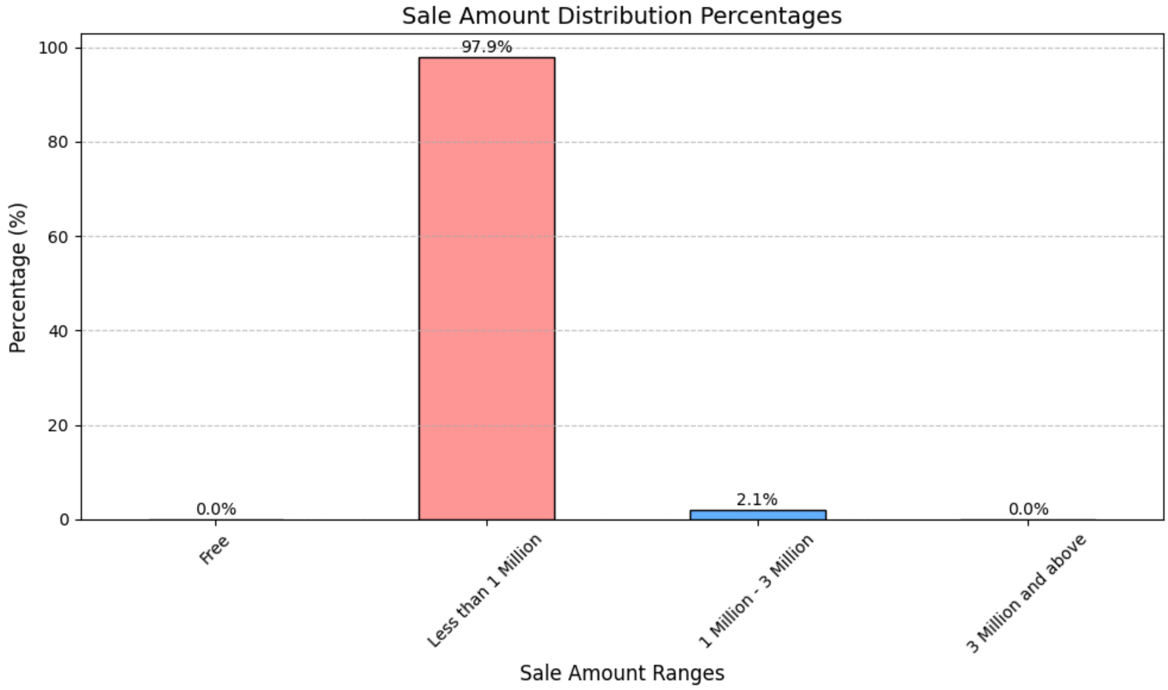
```
plt.figure(figsize=(12, 6))

sns.boxplot(data=data5[['Sale Amount', 'Assessed Value']])

plt.title(" Sale Amount, Assessed Value - Boxplot", fontsize=16)
```

```
plt.show()
```

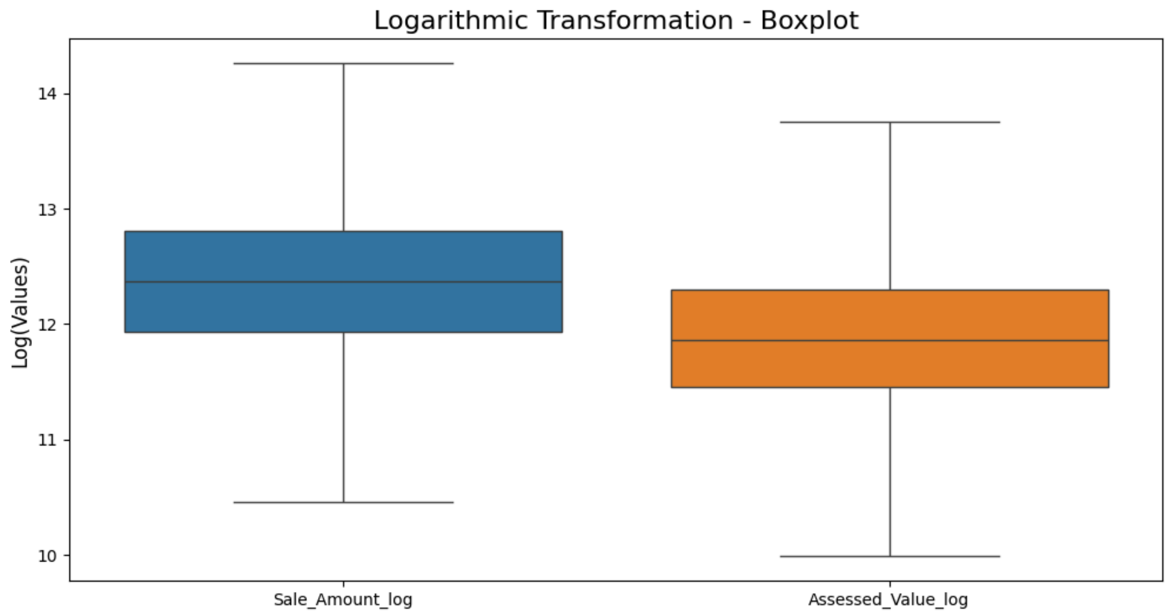


In [ ]:

```
bins = [0, 1, 1000000, 3000000, float('inf')]
labels = ['Free', 'Less than 1 Million', '1 Million - 3 Million', '3 Million and

category_counts = pd.cut(data5['Sale Amount'], bins=bins, labels=labels, right=F

plt.figure(figsize=(10, 6))
colors = ['#FFCC99', '#FF9999', '#66B3FF', '#99FF99']
category_counts.sort_index().plot(kind='bar', color=colors, edgecolor='black')

plt.title('Sale Amount Distribution Percentages', fontsize=14)
plt.xlabel('Sale Amount Ranges', fontsize=12)
plt.ylabel('Percentage (%)', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)

for i, value in enumerate(category_counts.sort_index()):
    plt.text(i, value + 1, f'{value:.1f}%', ha='center', fontsize=10)

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

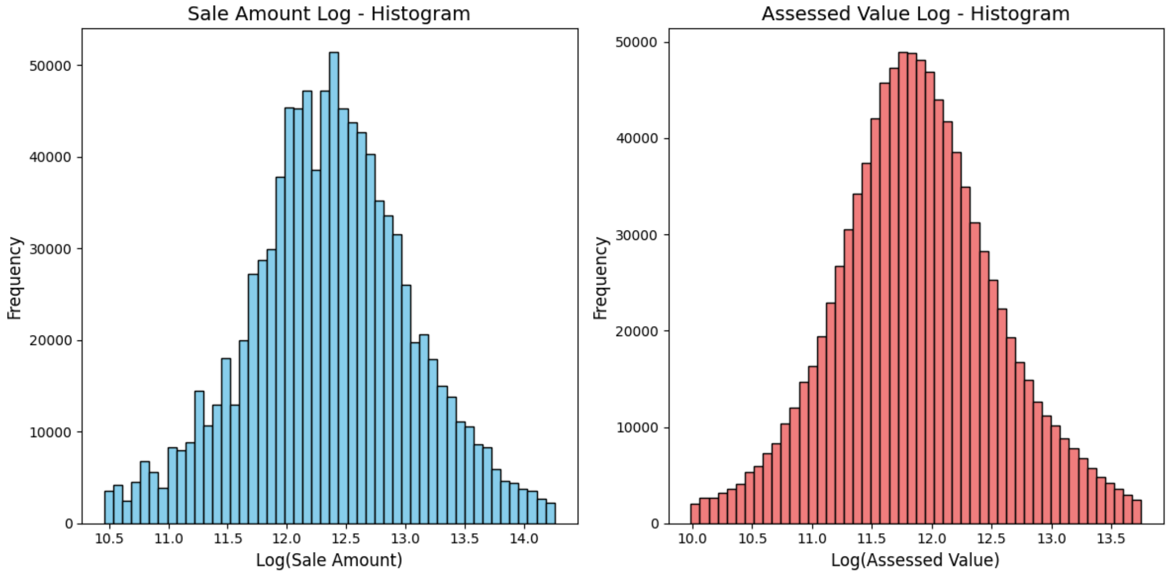


In [ ]:
```
plt.figure(figsize=(12, 6))

sns.boxplot(data=data5[['Sale_Amount_log', 'Assessed_Value_log']], whis = 3)

plt.title("Logarithmic Transformation - Boxplot", fontsize=16)
plt.ylabel("Log(Values)", fontsize=12)
plt.show()
```



In [ ]:
```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(data5['Sale_Amount_log'], bins=50, color='skyblue', edgecolor='black')
plt.title("Sale Amount Log - Histogram", fontsize=14)
plt.xlabel("Log(Sale Amount)", fontsize=12)
plt.ylabel("Frequency", fontsize=12)

plt.subplot(1, 2, 2)
plt.hist(data5['Assessed_Value_log'], bins=50, color='lightcoral', edgecolor='bl
plt.title("Assessed Value Log - Histogram", fontsize=14)
```

```
plt.xlabel("Log(Assessed Value)", fontsize=12)
plt.ylabel("Frequency", fontsize=12)

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



In [ ]:
```
log_summary = data5[['Sale_Amount_log', 'Assessed_Value_log']].describe()
print(log_summary)
```

```
       Sale_Amount_log  Assessed_Value_log
count    995169.000000       995169.000000
mean         12.363346           11.879414
std           0.690669            0.668185
min          10.460099            9.986495
25%          11.931642           11.451167
50%          12.367345           11.860839
75%          12.807655           12.297288
max          14.259761           13.749372
```

In [ ]:
```
data5.to_csv('final_data.csv')
```

Repeat Sales Method

In [18]:
```
RSM_data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Projects/Personal
RSM_data.drop(columns='Unnamed: 0',inplace=True)
```

In [19]:
```
RSM_data
```

Out[19]:

	Serial Number	List Year	Date Recorded	Town	Address	Assessed Value	Sale Amount	
0	60228	2006	2007-07-05	Bethel	10 HUNTINGTON COURT	120960.0	250000.0	0.48
1	60075	2006	2007-04-05	Essex	7 PRATT ST	143400.0	339500.0	0.42
2	60416	2006	2007-05-25	Newington	29 STERLING DR	221970.0	340000.0	0.65
3	60537	2006	2007-08-31	Branford	91 JEFFERSON WOODS	118800.0	210000.0	0.56
4	60421	2006	2007-05-08	Glastonbury	9 BOXWOOD LN	84000.0	174000.0	0.48
...	...	...	...	...	...	...	...	
995164	190234	2019	2020-07-20	Wilton	481 DANBURY RD	445200.0	410000.0	1.08
995165	19150	2019	2020-01-13	Newtown	22 WASHINGTON AVENUE	53640.0	122500.0	0.43
995166	190242	2019	2020-09-18	Weston	OLD HYDE ROAD	181440.0	150000.0	1.20
995167	19000067	2019	2020-05-19	New Hartford	LOT 2 DINGS RD	87955.0	35000.0	2.51
995168	190713	2019	2020-06-01	New Haven	1083 WHALLEY AV	262220.0	325000.0	0.80

995169 rows × 12 columns

Let us add the quarter of the year of each transaction recorded

We are going to use the 'Date Recorded'

In [20]:
```
RSM_data['Date Recorded'] = pd.to_datetime(RSM_data['Date Recorded'])
RSM_data['Year'] = RSM_data['Date Recorded'].dt.year
```

In [21]:
```
RSM_data.drop(columns='List Year',inplace=True)
```

In [22]:
```
RSM_data['Quarter'] = RSM_data['Date Recorded'].dt.quarter
```

In [23]:
```
RSM_data
```

Out[23]:

	Serial Number	Date Recorded	Town	Address	Assessed Value	Sale Amount	Sales Ratio
0	60228	2007-07-05	Bethel	10 HUNTINGTON COURT	120960.0	250000.0	0.483840
1	60075	2007-04-05	Essex	7 PRATT ST	143400.0	339500.0	0.422386
2	60416	2007-05-25	Newington	29 STERLING DR	221970.0	340000.0	0.652853
3	60537	2007-08-31	Branford	91 JEFFERSON WOODS	118800.0	210000.0	0.565714
4	60421	2007-05-08	Glastonbury	9 BOXWOOD LN	84000.0	174000.0	0.482759
...	...	...	...	...	...	...	...
995164	190234	2020-07-20	Wilton	481 DANBURY RD	445200.0	410000.0	1.085900
995165	19150	2020-01-13	Newtown	22 WASHINGTON AVENUE	53640.0	122500.0	0.437900
995166	190242	2020-09-18	Weston	OLD HYDE ROAD	181440.0	150000.0	1.209600
995167	19000067	2020-05-19	New Hartford	LOT 2 DINGS RD	87955.0	35000.0	2.513000
995168	190713	2020-06-01	New Haven	1083 WHALLEY AV	262220.0	325000.0	0.806800

995169 rows × 13 columns

Keep only properties sold more than one time

We will create a new dataset for based on the first and last listing years of repeated addresses.

New data was produced for price changes and repeat sales of real estate over time.

To diffrentiate between each unique property. We will create a new dataset based on the first and last listing years of repeated addresses with the same property and residential type.

In [24]:
```
duplicate_addresses_prop_res = RSM_data[RSM_data.duplicated(subset=['Address','P
property_data = RSM_data[['Address', 'Town','Property Type','Residential Type']]
```

In [25]:
```
duplicate_addresses_prop_res
```

Out[25]:

	Serial Number	Date Recorded	Town	Address	Assessed Value	Sale Amount	Sales Ratio
0	60228	2007-07-05	Bethel	10 HUNTINGTON COURT	120960.0	250000.0	0.483840
4	60421	2007-05-08	Glastonbury	9 BOXWOOD LN	84000.0	174000.0	0.482759
7	60082	2007-07-19	Marlborough	11 SACHEM DR	158900.0	255000.0	0.623137
9	60327	2007-07-13	Cromwell	8 WATCH HL CIR	97050.0	179900.0	0.539466
10	60354	2007-04-23	Newington	41 WEBSTER CT	85750.0	152000.0	0.564145
...	...	...	...	...	...	...	...
995150	190275	2020-08-03	Wilton	42 BORGLUM RD	253050.0	320000.0	0.790800
995152	190065	2019-12-26	Winchester	135 MAIN ST	89950.0	200000.0	0.449800
995155	190049	2020-07-02	Roxbury	42 WELTON RD	277200.0	305000.0	0.908900
995157	1910383	2020-06-16	Naugatuck	1152 NEW HAVEN RD	104050.0	245000.0	0.424694
995166	190242	2020-09-18	Weston	OLD HYDE ROAD	181440.0	150000.0	1.209600

301651 rows × 13 columns

In [26]:

```
first_year_data = duplicate_addresses_prop_res[duplicate_addresses_prop_res['Yea
last_year_data = duplicate_addresses_prop_res[duplicate_addresses_prop_res['Year

first_year_data = first_year_data[['Address', 'Year', 'Sale Amount','Sale_Amount
last_year_data = last_year_data[['Address', 'Year', 'Sale Amount','Sale_Amount_l

merged_data = pd.merge(first_year_data, last_year_data, on=['Address','Property

RSM = merged_data[merged_data['First Year'] != merged_data['Last Year']]
```

In [27]:

```
RSM
```

Out[27]:

	Address	First Year	First_Year_Sale_Amount	First_Year_Sale_Amount_log	Pro
0	10 HUNTINGTON COURT	2007	250000.0	12.429220	Resid
1	11 SACHEM DR	2007	255000.0	12.449023	Resid
2	2 MULLIGAN DR	2006	520000.0	13.161586	Resid
3	168 SUMMIT ST	2007	184000.0	12.122696	Resid
4	135 CLIFF ST	2006	110000.0	11.608245	Resid
...	...	...	...	...	
218614	155 NELLS ROCK RD	2020	103000.0	11.542494	V
218619	3 PARK AVE	2020	632385.0	13.357255	Comm
218622	193 MAIN ST	2019	265000.0	12.487489	Resid
218623	13 WEST ST	2020	175000.0	12.072547	Resid
218626	0 PLATT RD	2020	65000.0	11.082158	V

124408 rows × 11 columns

In [28]:

```
pd.DataFrame({
    'Count': RSM.count(),
    'Null': RSM.isnull().sum(),
    'Cardinality': RSM.nunique()
})
```

Out[28]:

	Count	Null	Cardinality
Address	124408	0	106445
First Year	124408	0	23
First_Year_Sale_Amount	124408	0	11147
First_Year_Sale_Amount_log	124408	0	11147
Property Type	124408	0	6
Residential Type	124408	0	6
First_Year_Quarter	124408	0	4
Last Year	124408	0	22
Last_Year_Sale_Amount	124408	0	8325
Last_Year_Sale_Amount_log	124408	0	8325
Last_Year_Quarter	124408	0	4

There is only **103795** properties sold more than once from the original **771931** properties (about **13.5%**)

Exploratory Data Analysis

It is done in Microsoft Power BI

Calculation method of Price Index : Preparing the training dataset

For the repeat sales method, only the **price change** and the **number of transactions** are included in the construction of the index.

It creates an **index sensitive to the market dynamics**, taking into account the **time distribution of transactions**.

Thus, each repeated sale (couple of transactions on the same property) is used to calculate a price change.

The index is then constructed on the basis of these individual transactions.

Linear Regresssion Model :

$$\log\left(\frac{P_{it}}{P_{i\tau}}\right) = \sum_{s} \beta_s D_{is} + \varepsilon_{i\tau}, \quad s = 1, \ldots, S$$

With

$$D_{is} = \begin{cases} 1 & \text{if } s = t \\ -1 & \text{if } s = \tau, \quad \text{with } t > \tau \\ 0 & \text{else} \end{cases}$$

Where:

- $P_{i\tau}$: Price of the property at the time τ, date of the first sale
- P_{it}: Price of the same property at the time t, date of the second sale
- β_s: Coefficient to estimate for the period s
- $\varepsilon_{i\tau}$: Error term
- S: Number of quarters contained in the study period

Where $\tau = t - 1$, price development is assimilated to average price movements on repeat sales observed between t and $t - 1$.

Once estimated, the coefficients β_s are used to construct the index on a base of 100 for the quarter t:

$$I_t = 100 \exp(\hat{\beta}_t - \hat{\beta}_\tau)$$

In [29]:
```
RSM
```

Out[29]:

	Address	First Year	First_Year_Sale_Amount	First_Year_Sale_Amount_log	Pro
0	10 HUNTINGTON COURT	2007	250000.0	12.429220	Resid
1	11 SACHEM DR	2007	255000.0	12.449023	Resid
2	2 MULLIGAN DR	2006	520000.0	13.161586	Resid
3	168 SUMMIT ST	2007	184000.0	12.122696	Resid
4	135 CLIFF ST	2006	110000.0	11.608245	Resid
...	...	...	...	...	
218614	155 NELLS ROCK RD	2020	103000.0	11.542494	V
218619	3 PARK AVE	2020	632385.0	13.357255	Comm
218622	193 MAIN ST	2019	265000.0	12.487489	Resid
218623	13 WEST ST	2020	175000.0	12.072547	Resid
218626	0 PLATT RD	2020	65000.0	11.082158	V

124408 rows × 11 columns

In [30]:
```
RSM_1 = RSM.copy()
```

In [31]:
```
RSM_1 = RSM_1.reset_index()
```

In [32]:
```
RSM_1.drop(columns='index',inplace=True)
```

In [33]:
```
RSM_1
```

Out[33]:

	Address	First Year	First_Year_Sale_Amount	First_Year_Sale_Amount_log	Pro
0	10 HUNTINGTON COURT	2007	250000.0	12.429220	Resid
1	11 SACHEM DR	2007	255000.0	12.449023	Resid
2	2 MULLIGAN DR	2006	520000.0	13.161586	Resid
3	168 SUMMIT ST	2007	184000.0	12.122696	Resid
4	135 CLIFF ST	2006	110000.0	11.608245	Resid
...	...	...	...	...	
124403	155 NELLS ROCK RD	2020	103000.0	11.542494	V
124404	3 PARK AVE	2020	632385.0	13.357255	Comm
124405	193 MAIN ST	2019	265000.0	12.487489	Resid
124406	13 WEST ST	2020	175000.0	12.072547	Resid
124407	0 PLATT RD	2020	65000.0	11.082158	V

124408 rows × 11 columns

In [34]:
```
RSM_1['First Year'].unique()
```

Out[34]:
```
array([2007, 2006, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2017,
       2018, 2019, 2020, 2001, 2004, 2005, 2016, 1999, 2021, 2022, 2002,
       2003], dtype=int32)
```

In [35]:
```
RSM_1['Last Year'].unique()
```

Out[35]:
```
array([2014, 2023, 2016, 2010, 2018, 2020, 2019, 2022, 2017, 2012, 2021,
       2013, 2007, 2015, 2008, 2011, 2009, 2002, 2004, 2003, 2006, 2005],
      dtype=int32)
```

In [36]:
```
years = list(map(str,range(2001,2023)))
quarters = ['Q1','Q2','Q3','Q4']
```

In [37]:
```
qr_yr_cl = [f"{yr} {q}" for yr, q in itertools.product(years, quarters)]
RSM_1[qr_yr_cl] = 0
```

In [38]:
```
RSM_1.head()
```

Out[38]:

	Address	First Year	First_Year_Sale_Amount	First_Year_Sale_Amount_log	Property Type
0	10 HUNTINGTON COURT	2007	250000.0	12.429220	Residential
1	11 SACHEM DR	2007	255000.0	12.449023	Residential
2	2 MULLIGAN DR	2006	520000.0	13.161586	Residential
3	168 SUMMIT ST	2007	184000.0	12.122696	Residential
4	135 CLIFF ST	2006	110000.0	11.608245	Residential

5 rows × 99 columns

In [39]:

```
RSM_1
```

Out[39]:

	Address	First Year	First_Year_Sale_Amount	First_Year_Sale_Amount_log	Pro
0	10 HUNTINGTON COURT	2007	250000.0	12.429220	Resid
1	11 SACHEM DR	2007	255000.0	12.449023	Resid
2	2 MULLIGAN DR	2006	520000.0	13.161586	Resid
3	168 SUMMIT ST	2007	184000.0	12.122696	Resid
4	135 CLIFF ST	2006	110000.0	11.608245	Resid
...	...	...	...	...	
124403	155 NELLS ROCK RD	2020	103000.0	11.542494	V
124404	3 PARK AVE	2020	632385.0	13.357255	Comm
124405	193 MAIN ST	2019	265000.0	12.487489	Resid
124406	13 WEST ST	2020	175000.0	12.072547	Resid
124407	0 PLATT RD	2020	65000.0	11.082158	V

124408 rows × 99 columns

In [40]:
```
qr_yr_df = pd.DataFrame({
    'Column': qr_yr_cl,
    'Year': [int(c[:4]) for c in qr_yr_cl],
    'Quarter': [int(c[-1]) for c in qr_yr_cl]
})
```

In [41]:
```
for _, row in qr_yr_df.iterrows():
    col, year, quarter = row['Column'], row['Year'], row['Quarter']

    RSM_1.loc[(RSM_1['First Year'] == year) & (RSM_1['First_Year_Quarter'] == qu
    RSM_1.loc[(RSM_1['Last Year'] == year) & (RSM_1['Last_Year_Quarter'] == quar
```

In [42]:
```
RSM_1
```

Out[42]:

	Address	First Year	First_Year_Sale_Amount	First_Year_Sale_Amount_log	Pro
0	10 HUNTINGTON COURT	2007	250000.0	12.429220	Resid
1	11 SACHEM DR	2007	255000.0	12.449023	Resid
2	2 MULLIGAN DR	2006	520000.0	13.161586	Resid
3	168 SUMMIT ST	2007	184000.0	12.122696	Resid
4	135 CLIFF ST	2006	110000.0	11.608245	Resid
...	...	...	...	...	
124403	155 NELLS ROCK RD	2020	103000.0	11.542494	V
124404	3 PARK AVE	2020	632385.0	13.357255	Comm
124405	193 MAIN ST	2019	265000.0	12.487489	Resid
124406	13 WEST ST	2020	175000.0	12.072547	Resid
124407	0 PLATT RD	2020	65000.0	11.082158	V

124408 rows × 99 columns

In [43]:
```
RSM_1['log(Last_Year_Sale_Amount/         First_Year_Sale_Amount)']=RSM_1['Last_Ye
```

In [44]:
```
RSM_1
```

Out[44]:

	Address	First Year	First_Year_Sale_Amount	First_Year_Sale_Amount_log	Pro
0	10 HUNTINGTON COURT	2007	250000.0	12.429220	Resid
1	11 SACHEM DR	2007	255000.0	12.449023	Resid
2	2 MULLIGAN DR	2006	520000.0	13.161586	Resid
3	168 SUMMIT ST	2007	184000.0	12.122696	Resid
4	135 CLIFF ST	2006	110000.0	11.608245	Resid
...	...	...	...	...	
124403	155 NELLS ROCK RD	2020	103000.0	11.542494	V
124404	3 PARK AVE	2020	632385.0	13.357255	Comm
124405	193 MAIN ST	2019	265000.0	12.487489	Resid
124406	13 WEST ST	2020	175000.0	12.072547	Resid
124407	0 PLATT RD	2020	65000.0	11.082158	V

124408 rows × 100 columns

In [45]:

```
RSM_1.to_csv('training_data.csv')
```

Calculation method of Price Index : Fitting the model

Linear Regresssion Model :

$$\log\left(\frac{P_{it}}{P_{i\tau}}\right) = \sum_s \beta_s D_{is} + \varepsilon_{i\tau}, \quad s = 1, \ldots, S$$

With

$$D_{is} = \begin{cases} 1 & \text{if } s = t \\ -1 & \text{if } s = \tau, \quad \text{with } t > \tau \\ 0 & \text{else} \end{cases}$$

Where:

- $P_{i\tau}$: Price of the property at the time τ, date of the first sale
- P_{it}: Price of the same property at the time t, date of the second sale
- β_s: Coefficient to estimate for the period s
- $\varepsilon_{i\tau}$: Error term
- S: Number of quarters contained in the study period

Where $\tau = t - 1$, price development is assimilated to average price movements on repeat sales observed between t and $t - 1$.

Once estimated, the coefficients β_s are used to construct the index on a base of 100 for the quarter t:

$$I_t = 100 \exp(\hat{\beta}_t - \hat{\beta}_\tau)$$

In [46]:
```
training_data = RSM_1.copy()
```

In [47]:
```
training_data.head()
```

Out[47]:

	Address	First Year	First_Year_Sale_Amount	First_Year_Sale_Amount_log	Property Type
0	10 HUNTINGTON COURT	2007	250000.0	12.429220	Residential
1	11 SACHEM DR	2007	255000.0	12.449023	Residential
2	2 MULLIGAN DR	2006	520000.0	13.161586	Residential
3	168 SUMMIT ST	2007	184000.0	12.122696	Residential
4	135 CLIFF ST	2006	110000.0	11.608245	Residential

5 rows × 100 columns

In [48]:
```
training_data['Property Type'].unique()
```

Out[48]:
```
array(['Residential', 'Commercial', 'Vacant Land', 'Apartments',
       'Industrial', 'Public Utility'], dtype=object)
```

In [49]:
```
training_data['Residential Type'].unique()
```

Out[49]:
```
array(['Condo', 'Two Family', 'Single Family', 'Four Family',
       'Three Family', 'Non Residential'], dtype=object)
```

In [50]:
```
global_index = training_data.drop(columns=['Address','First Year','First_Year_Sa

prop_index_res = training_data[training_data['Property Type']=='Residential'].dr

prop_index_com = training_data[training_data['Property Type']=='Commercial'].drc
```

```
prop_index_vl = training_data[training_data['Property Type']=='Vacant Land'].drc
prop_index_app = training_data[training_data['Property Type']=='Apartments'].drc
prop_index_in = training_data[training_data['Property Type']=='Industrial'].drop
prop_index_pu = training_data[training_data['Property Type']=='Public Utility'].
res_index_co = training_data[training_data['Residential Type']=='Condo'].drop(cc
res_index_tf = training_data[training_data['Residential Type']=='Two Family'].dr
res_index_sf = training_data[training_data['Residential Type']=='Single Family']
res_index_ff = training_data[training_data['Residential Type']=='Four Family'].d
res_index_thf = training_data[training_data['Residential Type']=='Three Family']
```

In [51]:
```
global_index
```

Out[51]:

	2001 Q1	2001 Q2	2001 Q3	2001 Q4	2002 Q1	2002 Q2	2002 Q3	2002 Q4	2003 Q1	2003 Q2	...	2020 Q4	20. (
0	0	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
124403	0	0	0	0	0	0	0	0	0	0	...	0	
124404	0	0	0	0	0	0	0	0	0	0	...	0	
124405	0	0	0	0	0	0	0	0	0	0	...	0	
124406	0	0	0	0	0	0	0	0	0	0	...	0	
124407	0	0	0	0	0	0	0	0	0	0	...	0	

124408 rows × 89 columns

In [52]:
```
y = global_index['log(Last_Year_Sale_Amount/\tFirst_Year_Sale_Amount)']
```

In [53]:
```
y
```

Out[53]:

	log(Last_Year_Sale_Amount/\tFirst_Year_Sale_Amount)
0	-0.573407
1	0.195308
2	-0.080043
3	-0.224502
4	-0.850321
...	...
124403	0.638266
124404	-1.102388
124405	0.306373
124406	-0.847290
124407	0.430778

124408 rows × 1 columns

dtype: float64

In [54]:

```
qr_yr_df['Column']
```

Out[54]:

	Column
0	2001 Q1
1	2001 Q2
2	2001 Q3
3	2001 Q4
4	2002 Q1
...	...
83	2021 Q4
84	2022 Q1
85	2022 Q2
86	2022 Q3
87	2022 Q4

88 rows × 1 columns

dtype: object

In [55]:
```
design = MS(qr_yr_df['Column'])
design = design.fit(global_index)
X = design.transform(global_index)
```

In [56]:
```
X
```

Out[56]:

	intercept	2001 Q1	2001 Q2	2001 Q3	2001 Q4	2002 Q1	2002 Q2	2002 Q3	2002 Q4	2003 Q1	...	2020 Q3
0	1.0	0	0	0	0	0	0	0	0	0	...	0
1	1.0	0	0	0	0	0	0	0	0	0	...	0
2	1.0	0	0	0	0	0	0	0	0	0	...	0
3	1.0	0	0	0	0	0	0	0	0	0	...	0
4	1.0	0	0	0	0	0	0	0	0	0	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
124403	1.0	0	0	0	0	0	0	0	0	0	...	-1
124404	1.0	0	0	0	0	0	0	0	0	0	...	-1
124405	1.0	0	0	0	0	0	0	0	0	0	...	0
124406	1.0	0	0	0	0	0	0	0	0	0	...	0
124407	1.0	0	0	0	0	0	0	0	0	0	...	0

124408 rows × 89 columns

In [57]:
```
model = sm.OLS(y, X)
results = model.fit()
```

In [58]:
```
summarize(results)
```

Out[58]:

	coef	std err	t	P>|t|
intercept	2.244000e-01	3.000000e-03	83.526	0.000
2001 Q1	-6.511000e-16	6.050000e-17	-10.763	0.000
2001 Q2	5.114000e-16	3.500000e-17	14.596	0.000
2001 Q3	3.900000e-03	3.680000e-01	0.011	0.992
2001 Q4	-5.970000e-02	1.400000e-02	-4.351	0.000
...	...	...	...	...
2021 Q4	-1.670000e-01	1.300000e-02	-13.031	0.000
2022 Q1	-7.780000e-02	1.400000e-02	-5.558	0.000
2022 Q2	-4.190000e-02	1.300000e-02	-3.231	0.001
2022 Q3	-5.390000e-02	1.300000e-02	-4.247	0.000
2022 Q4	-9.870000e-02	1.400000e-02	-7.219	0.000

89 rows × 4 columns

In [59]:

```
print(results.params)
```

```
intercept    2.243567e-01
2001 Q1     -6.511352e-16
2001 Q2      5.114146e-16
2001 Q3      3.904378e-03
2001 Q4     -5.971770e-02
                 ...
2021 Q4     -1.670110e-01
2022 Q1     -7.778264e-02
2022 Q2     -4.190409e-02
2022 Q3     -5.386777e-02
2022 Q4     -9.874669e-02
Length: 89, dtype: float64
```

In [60]:

```
beta = pd.Series(results.params)[1:]
```

In [61]:

```
beta
```

Out[61]:

	0
2001 Q1	-6.511352e-16
2001 Q2	5.114146e-16
2001 Q3	3.904378e-03
2001 Q4	-5.971770e-02
2002 Q1	-1.481546e-01
...	...
2021 Q4	-1.670110e-01
2022 Q1	-7.778264e-02
2022 Q2	-4.190409e-02
2022 Q3	-5.386777e-02
2022 Q4	-9.874669e-02

88 rows × 1 columns

dtype: float64

In [62]:

```
beta_tau = beta[0]

index_global = 100 * np.exp(beta - beta_tau)
```

```
<ipython-input-62-d259f579a4e1>:1: FutureWarning: Series.__getitem__ treating key
s as positions is deprecated. In a future version, integer keys will always be tr
eated as labels (consistent with DataFrame behavior). To access a value by positi
on, use `ser.iloc[pos]`
  beta_tau = beta[0]
```

In [63]:

```
index_global.rename({'0':'Property Index'}, inplace=True)
```

In [64]:

```
index_global.name = 'Property Index'
```

In [65]:

```
index_global
```

Out[65]:

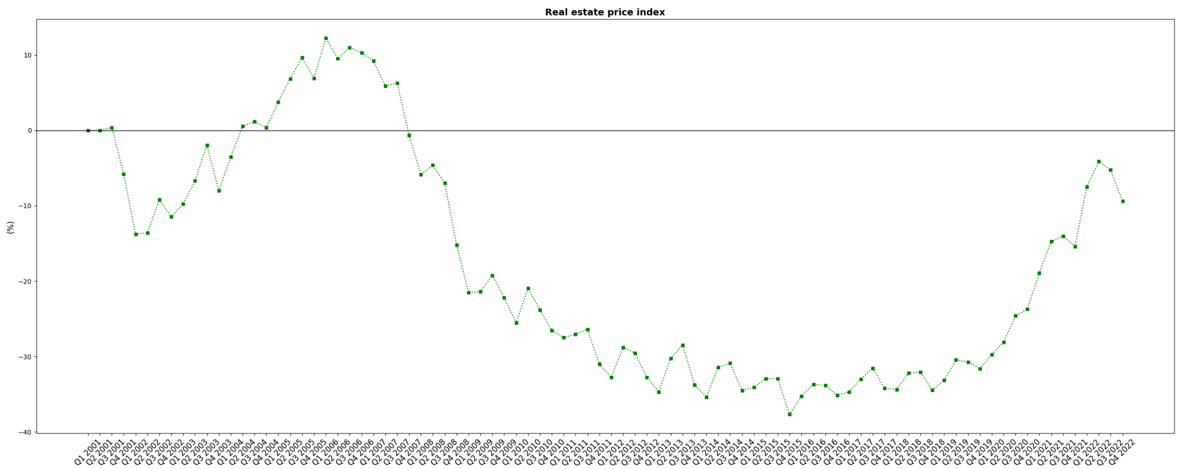
	Property Index
2001 Q1	100.000000
2001 Q2	100.000000
2001 Q3	100.391201
2001 Q4	94.203043
2002 Q1	86.229783
...	...
2021 Q4	84.619031
2022 Q1	92.516550
2022 Q2	95.896175
2022 Q3	94.755739
2022 Q4	90.597217

88 rows × 1 columns

dtype: float64

In [66]:

```
quarters = [f"Q{q} {y}" for y in range(2001, 2023) for q in range(1, 5)]
quarters = quarters[:88]

y = index_global.values
y = y - y[0]

fig, ax = plt.subplots(figsize=(25, 10))
ax.plot(quarters, y, marker='s', markersize=5, color='green', linestyle="dotted"

ax.axhline(0, color='black', linewidth=1)
ax.set_ylabel('(%)', fontsize=12)
ax.set_title('Real estate price index', fontsize=14, weight='bold')
ax.set_xticks(range(len(quarters)))
ax.set_xticklabels(quarters, rotation=45, fontsize=12)
ax.set_ylim()

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



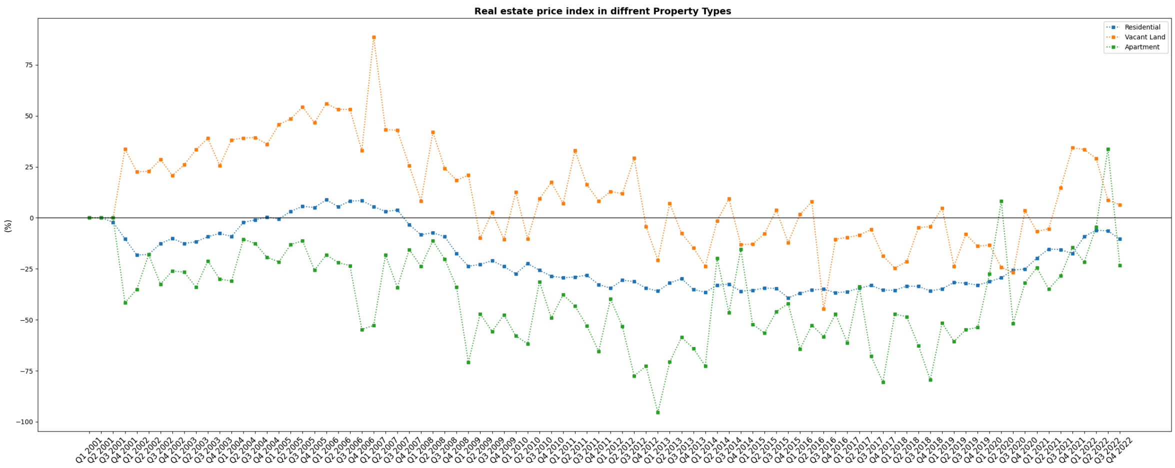
In [67]:
```
def index (data, index_name):
  y = data['log(Last_Year_Sale_Amount/\tFirst_Year_Sale_Amount)']
  design = MS(qr_yr_df['Column'])
  design = design.fit(data)
  X = design.transform(data)
  model = sm.OLS(y, X)
  results = model.fit()
  beta = pd.Series(results.params)[1:]

  beta_tau = beta.iloc[0]
  index = 100 * np.exp(beta - beta_tau)
  index.name = index_name

  return index
```

In [68]:
```
index_prop_res = index (prop_index_res, 'Residential Property Index')
index_prop_com = index (prop_index_com, 'Commercial Property Index')
index_prop_vl = index (prop_index_vl, 'Vacant Land Property Index')
index_prop_app = index (prop_index_app, 'Apartement Property Index')
index_prop_in = index (prop_index_in, 'Industrial Property Index')
index_prop_pu = index (prop_index_pu, 'Public Utility Property Index')
```

In [69]:
```
index_prop = pd.concat([index_prop_res, index_prop_vl,index_prop_app], axis=1)
```

In [70]:
```
quarters = [f"Q{q} {y}" for y in range(2001, 2023) for q in range(1, 5)]
quarters = quarters[:88]

y = index_prop.values
y = y - y[0]

fig, ax = plt.subplots(figsize=(25, 10))

ax.plot(quarters, y[:, 0], marker='s', markersize=5, linestyle="dotted", label='
ax.plot(quarters, y[:, 1], marker='s', markersize=5, linestyle="dotted", label='
ax.plot(quarters, y[:, 2], marker='s', markersize=5, linestyle="dotted", label='


ax.axhline(0, color='black', linewidth=1)
ax.set_ylabel('(%)', fontsize=12)
ax.set_title('Real estate price index in diffrent Property Types ', fontsize=14,
ax.set_xticks(range(len(quarters)))
ax.set_xticklabels(quarters, rotation=45, fontsize=12)
ax.set_ylim()
```

```python
ax.legend()

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



Forcast the price index in the future

In [71]:
```python
index_global
```

Out[71]:

	Property Index
2001 Q1	100.000000
2001 Q2	100.000000
2001 Q3	100.391201
2001 Q4	94.203043
2002 Q1	86.229783
...	...
2021 Q4	84.619031
2022 Q1	92.516550
2022 Q2	95.896175
2022 Q3	94.755739
2022 Q4	90.597217

88 rows × 1 columns

dtype: float64

In [72]:
```python
#!pip install prophet --upgrade
```

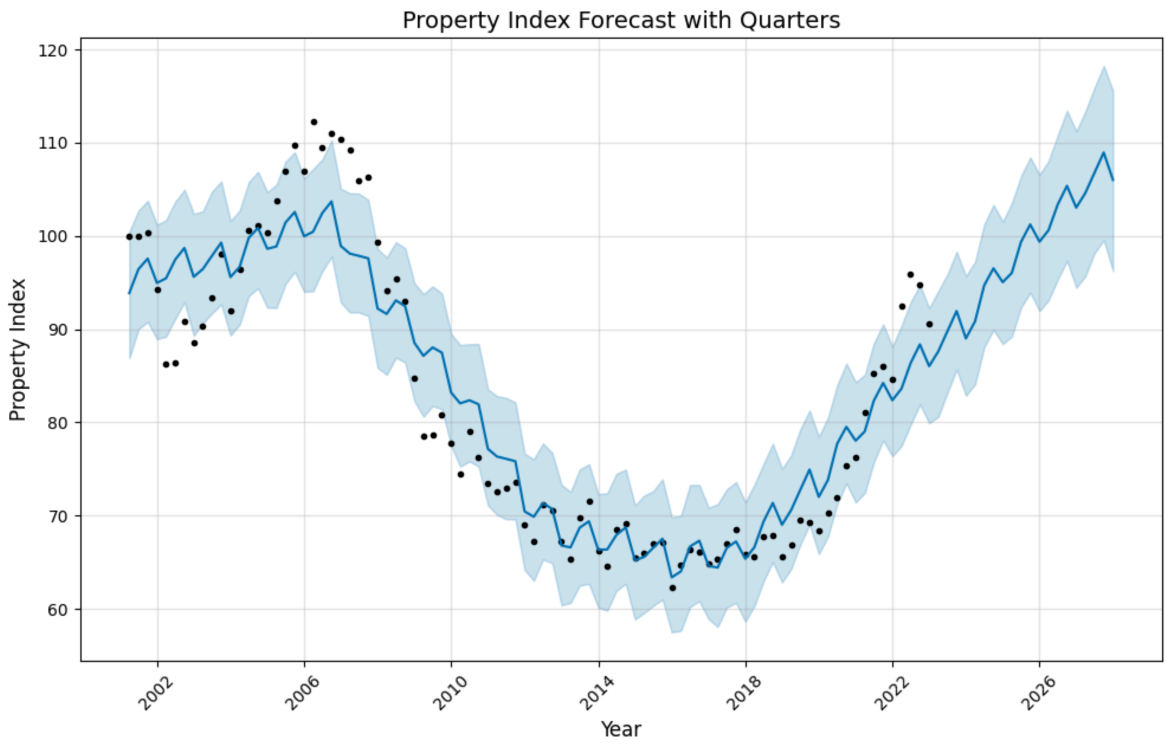
In [73]:
```python
df = index_global.copy().reset_index()
df.columns = ['quarter', 'y']

df[['year', 'q']] = df['quarter'].str.extract(r'(\d{4})\s*Q([1-4])')
```

```
quarter_end_month = {'1': '03-31', '2': '06-30', '3': '09-30', '4': '12-31'}
df['ds'] = df.apply(lambda row: f"{row['year']}-{quarter_end_month[row['q']]}",
df['ds'] = pd.to_datetime(df['ds'])

df = df[['ds', 'y']]
```

In [75]:
```
model = Prophet(changepoint_prior_scale=0.05, seasonality_prior_scale=10)
model.fit(df)
future = model.make_future_dataframe(periods=20, freq='Q')
forecast = model.predict(future)
```

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=Tr
ue to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True
to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/t4bk2lww.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/69readah.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/s
tan_model/prophet_model.bin', 'random', 'seed=18444', 'data', 'file=/tmp/tmps0mtr
kva/t4bk2lww.json', 'init=/tmp/tmps0mtrkva/69readah.json', 'output', 'file=/tmp/t
mp003zrarn/prophet_modelkc0a3bwl/prophet_model-20250421111305.csv', 'method=optim
ize', 'algorithm=newton', 'iter=10000']
11:13:05 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:13:06 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarnin
g: 'Q' is deprecated and will be removed in a future version, please use 'QE' ins
tead.
  dates = pd.date_range(
```

In [76]:
```
fig = model.plot(forecast)

plt.xticks(rotation=45)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Property Index', fontsize=12)
plt.title('Property Index Forecast with Quarters', fontsize=14)

plt.show()
```



Automate Data transformation for training and Modeling

In [77]:
```
years = list(map(str,range(2001,2023)))
quarters = ['Q1','Q2','Q3','Q4']

qr_yr_cl = [f"{yr} {q}" for yr, q in itertools.product(years, quarters)]
RSM_1[qr_yr_cl] = 0

qr_yr_df = pd.DataFrame({
    'Column': qr_yr_cl,
    'Year': [int(c[:4]) for c in qr_yr_cl],
    'Quarter': [int(c[-1]) for c in qr_yr_cl]
})
```

In [78]:
```
def transform_data(RSM_data):
  RSM_data['Date Recorded'] = pd.to_datetime(RSM_data['Date Recorded'])
  RSM_data['Year'] = RSM_data['Date Recorded'].dt.year
  RSM_data['Quarter'] = RSM_data['Date Recorded'].dt.quarter
  RSM_data.drop(columns='List Year',inplace=True)

  duplicate_addresses_prop_res = RSM_data[RSM_data.duplicated(subset=['Address',
  property_data = RSM_data[['Address', 'Town','Property Type','Residential Type'

  first_year_data = duplicate_addresses_prop_res[duplicate_addresses_prop_res['Y
  last_year_data = duplicate_addresses_prop_res[duplicate_addresses_prop_res['Ye

  first_year_data = first_year_data[['Address', 'Year', 'Sale Amount','Sale_Amou
  last_year_data = last_year_data[['Address', 'Year', 'Sale Amount','Sale_Amount

  merged_data = pd.merge(first_year_data, last_year_data, on=['Address','Propert

  RSM = merged_data[merged_data['First Year'] != merged_data['Last Year']]
```

```
  RSM_1 = RSM.copy()
  RSM_1 = RSM_1.reset_index()
  RSM_1.drop(columns='index',inplace=True)

  years = list(map(str,range(2001,2023)))
  quarters = ['Q1','Q2','Q3','Q4']

  qr_yr_cl = [f"{yr} {q}" for yr, q in itertools.product(years, quarters)]
  RSM_1[qr_yr_cl] = 0

  qr_yr_df = pd.DataFrame({
      'Column': qr_yr_cl,
      'Year': [int(c[:4]) for c in qr_yr_cl],
      'Quarter': [int(c[-1]) for c in qr_yr_cl]
  })

  for _, row in qr_yr_df.iterrows():
    col, year, quarter = row['Column'], row['Year'], row['Quarter']

    RSM_1.loc[(RSM_1['First Year'] == year) & (RSM_1['First_Year_Quarter'] == qu
    RSM_1.loc[(RSM_1['Last Year'] == year) & (RSM_1['Last_Year_Quarter'] == quar

  RSM_1['log(Last_Year_Sale_Amount/     First_Year_Sale_Amount)']=RSM_1['Last_Ye

  return RSM_1
```

In [79]:
```
RSM_data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Projects/Personal
```

In [80]:
```
RSM_1 = transform_data(RSM_data)
```

In [87]:
```
def index (data, index_name):
  years = list(map(str,range(2001,2023)))
  quarters = ['Q1','Q2','Q3','Q4']

  qr_yr_cl = [f"{yr} {q}" for yr, q in itertools.product(years, quarters)]

  qr_yr_df = pd.DataFrame({
      'Column': qr_yr_cl,
      'Year': [int(c[:4]) for c in qr_yr_cl],
      'Quarter': [int(c[-1]) for c in qr_yr_cl]
  })

  y = data['log(Last_Year_Sale_Amount/\tFirst_Year_Sale_Amount)']
  design = MS(qr_yr_df['Column'])
  design = design.fit(data)
  X = design.transform(data)
  model = sm.OLS(y, X)
  results = model.fit()
  beta = pd.Series(results.params)[1:]

  beta_tau = beta.iloc[0]
  index = 100 * np.exp(beta - beta_tau)
  index.name = index_name

  return index, beta
```

In [88]:
```
training_data = RSM_1.copy()

global_index = training_data.drop(columns=['Address','First Year','First_Year_Sa
```

```
prop_index_res = training_data[training_data['Property Type']=='Residential'].dr

prop_index_com = training_data[training_data['Property Type']=='Commercial'].drc

prop_index_vl = training_data[training_data['Property Type']=='Vacant Land'].drc

prop_index_app = training_data[training_data['Property Type']=='Apartments'].drc

prop_index_in = training_data[training_data['Property Type']=='Industrial'].drop

prop_index_pu = training_data[training_data['Property Type']=='Public Utility'].

res_index_co = training_data[training_data['Residential Type']=='Condo'].drop(cc

res_index_tf = training_data[training_data['Residential Type']=='Two Family'].dr

res_index_sf = training_data[training_data['Residential Type']=='Single Family']

res_index_ff = training_data[training_data['Residential Type']=='Four Family'].d

res_index_thf = training_data[training_data['Residential Type']=='Three Family']
```

In [90]:
```
index_prop_res = index (prop_index_res, 'Residential Property Index')[0]
index_prop_com = index (prop_index_com, 'Commercial Property Index')[0]
index_prop_vl = index (prop_index_vl, 'Vacant Land Property Index')[0]
index_prop_app = index (prop_index_app, 'Apartement Property Index')[0]
index_prop_in = index (prop_index_in, 'Industrial Property Index')[0]
index_prop_pu = index (prop_index_pu, 'Public Utility Property Index')[0]
index_global = index (global_index, 'Global Property index')[0]
```

In [91]:
```
beta_prop_res = index (prop_index_res, 'Residential Property Index')[1]
beta_prop_com = index (prop_index_com, 'Commercial Property Index')[1]
beta_prop_vl = index (prop_index_vl, 'Vacant Land Property Index')[1]
beta_prop_app = index (prop_index_app, 'Apartement Property Index')[1]
beta_prop_in = index (prop_index_in, 'Industrial Property Index')[1]
beta_prop_pu = index (prop_index_pu, 'Public Utility Property Index')[1]
beta_global = index (global_index, 'Global Property index')[1]
```

In [84]:
```
def forecast(index):
  model = Prophet(changepoint_prior_scale=0.05, seasonality_prior_scale=10)
  model.fit(df)
  future = model.make_future_dataframe(periods=20, freq='Q')
  forecast = model.predict(future)
  return forecast, model
```

In [85]:
```
forecast_global = forecast(index_global)[0]
model_global = forecast(index_global)[1]
```

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=Tr
ue to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True
to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/yac9imti.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/te6vob4e.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/s
tan_model/prophet_model.bin', 'random', 'seed=30280', 'data', 'file=/tmp/tmps0mtr
kva/yac9imti.json', 'init=/tmp/tmps0mtrkva/te6vob4e.json', 'output', 'file=/tmp/t
mp003zrarn/prophet_model9ycfhxa6/prophet_model-20250421111332.csv', 'method=optim
ize', 'algorithm=newton', 'iter=10000']
11:13:32 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:13:32 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarnin
g: 'Q' is deprecated and will be removed in a future version, please use 'QE' ins
tead.
  dates = pd.date_range(
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=Tr
ue to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True
to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/q1h52r3f.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/msx8119b.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/s
tan_model/prophet_model.bin', 'random', 'seed=16679', 'data', 'file=/tmp/tmps0mtr
kva/q1h52r3f.json', 'init=/tmp/tmps0mtrkva/msx8119b.json', 'output', 'file=/tmp/t
mp003zrarn/prophet_modelv9dbqp_i/prophet_model-20250421111332.csv', 'method=optim
ize', 'algorithm=newton', 'iter=10000']
11:13:32 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:13:33 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarnin
g: 'Q' is deprecated and will be removed in a future version, please use 'QE' ins
tead.
  dates = pd.date_range(
```

In [93]:

```
forecast_prop_res = forecast (index_prop_res)[0]
forecast_prop_com = forecast (index_prop_com)[0]
forecast_prop_vl = forecast (index_prop_vl)[0]
forecast_prop_app = forecast (index_prop_app)[0]
forecast_prop_in = forecast (index_prop_in)[0]
forecast_prop_pu = forecast (index_prop_pu)[0]
forecast_global = forecast (index_global)[0]
```

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=Tr
ue to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True
to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/4y_6zd91.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/juy0j2re.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/s
tan_model/prophet_model.bin', 'random', 'seed=2783', 'data', 'file=/tmp/tmps0mtrk
va/4y_6zd91.json', 'init=/tmp/tmps0mtrkva/juy0j2re.json', 'output', 'file=/tmp/tm
p003zrarn/prophet_modeljmd082b0/prophet_model-20250421111945.csv', 'method=optimi
ze', 'algorithm=newton', 'iter=10000']
11:19:45 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:19:46 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarnin
g: 'Q' is deprecated and will be removed in a future version, please use 'QE' ins
tead.
  dates = pd.date_range(
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=Tr
ue to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True
to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/j5_w1mu9.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/z72k4bss.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/s
tan_model/prophet_model.bin', 'random', 'seed=89348', 'data', 'file=/tmp/tmps0mtr
kva/j5_w1mu9.json', 'init=/tmp/tmps0mtrkva/z72k4bss.json', 'output', 'file=/tmp/t
mp003zrarn/prophet_modell9uvq30f/prophet_model-20250421111946.csv', 'method=optim
ize', 'algorithm=newton', 'iter=10000']
11:19:46 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:19:46 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarnin
g: 'Q' is deprecated and will be removed in a future version, please use 'QE' ins
tead.
  dates = pd.date_range(
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=Tr
ue to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True
to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/r06t93uz.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/t67b8pwk.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/s
tan_model/prophet_model.bin', 'random', 'seed=14064', 'data', 'file=/tmp/tmps0mtr
kva/r06t93uz.json', 'init=/tmp/tmps0mtrkva/t67b8pwk.json', 'output', 'file=/tmp/t
mp003zrarn/prophet_modeleme7a8qj/prophet_model-20250421111946.csv', 'method=optim
ize', 'algorithm=newton', 'iter=10000']
11:19:46 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:19:47 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarnin
```

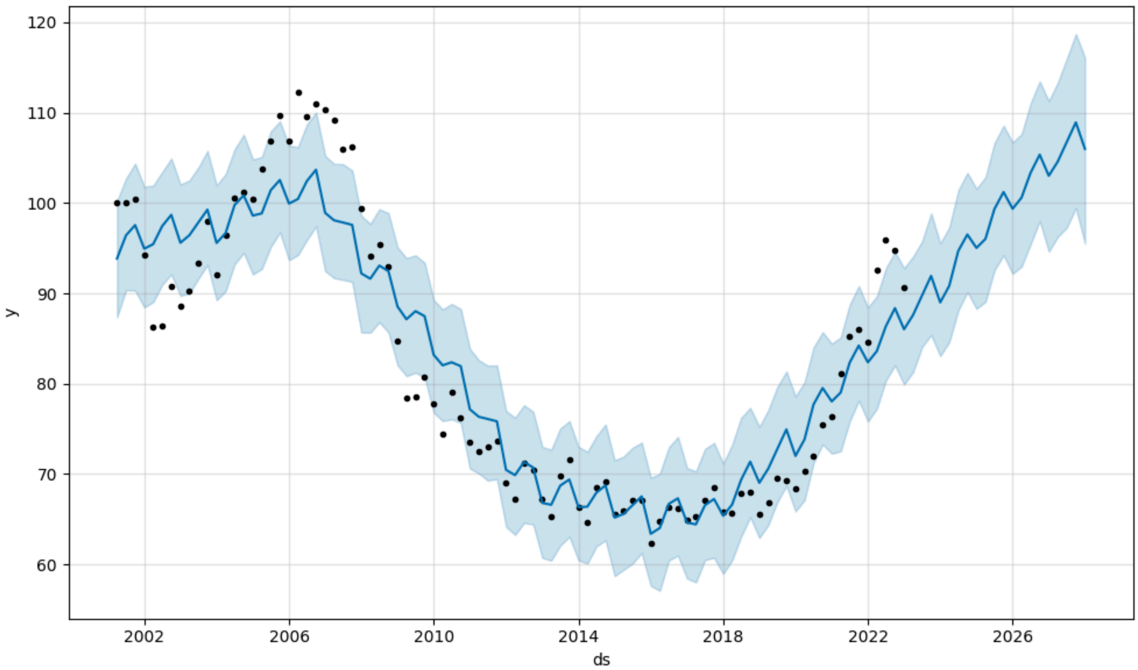
```
g: 'Q' is deprecated and will be removed in a future version, please use 'QE' ins
tead.
  dates = pd.date_range(
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=Tr
ue to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True
to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/8hz0wu7u.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/72uxdall.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/s
tan_model/prophet_model.bin', 'random', 'seed=67273', 'data', 'file=/tmp/tmps0mtr
kva/8hz0wu7u.json', 'init=/tmp/tmps0mtrkva/72uxdall.json', 'output', 'file=/tmp/t
mp003zrarn/prophet_model2n96k4_0/prophet_model-20250421111947.csv', 'method=optim
ize', 'algorithm=newton', 'iter=10000']
11:19:47 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:19:47 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarnin
g: 'Q' is deprecated and will be removed in a future version, please use 'QE' ins
tead.
  dates = pd.date_range(
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=Tr
ue to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True
to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/kzwuaj1i.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/cen_v7iz.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/s
tan_model/prophet_model.bin', 'random', 'seed=12508', 'data', 'file=/tmp/tmps0mtr
kva/kzwuaj1i.json', 'init=/tmp/tmps0mtrkva/cen_v7iz.json', 'output', 'file=/tmp/t
mp003zrarn/prophet_modeleqto2yl9/prophet_model-20250421111948.csv', 'method=optim
ize', 'algorithm=newton', 'iter=10000']
11:19:48 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:19:48 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarnin
g: 'Q' is deprecated and will be removed in a future version, please use 'QE' ins
tead.
  dates = pd.date_range(
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=Tr
ue to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True
to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/i5hgill6.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/5x1k0g7q.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/s
tan_model/prophet_model.bin', 'random', 'seed=37210', 'data', 'file=/tmp/tmps0mtr
kva/i5hgill6.json', 'init=/tmp/tmps0mtrkva/5x1k0g7q.json', 'output', 'file=/tmp/t
mp003zrarn/prophet_model03eeggd8/prophet_model-20250421111948.csv', 'method=optim
ize', 'algorithm=newton', 'iter=10000']
11:19:48 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
```

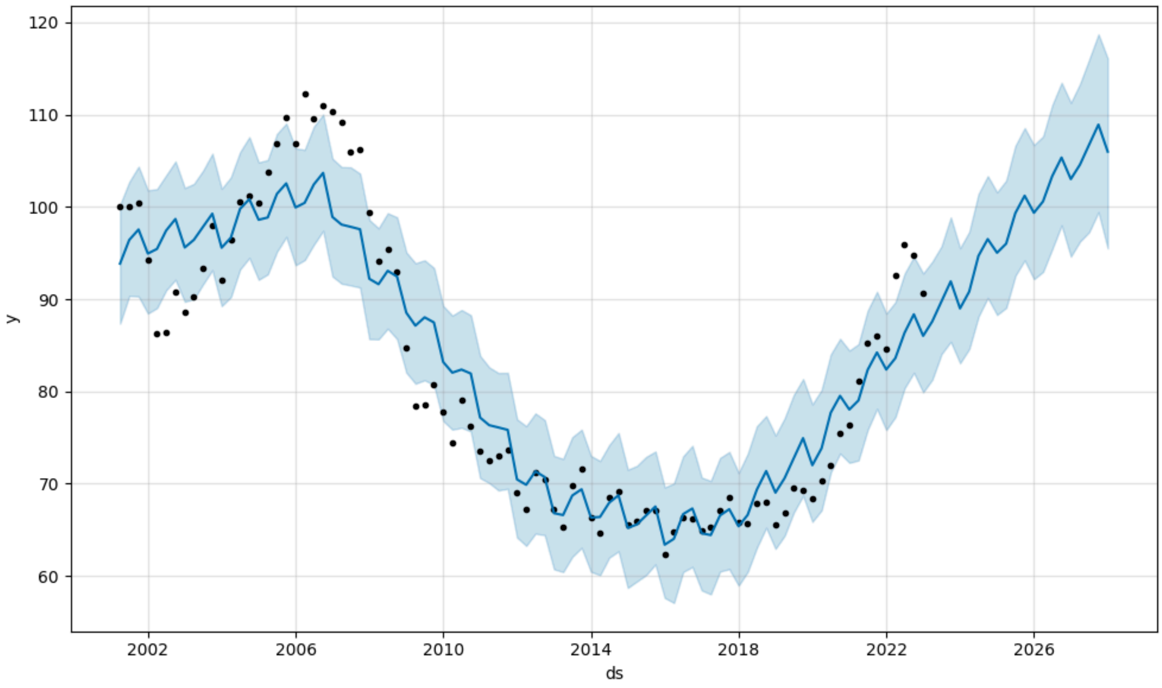
```
11:19:49 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarnin
g: 'Q' is deprecated and will be removed in a future version, please use 'QE' ins
tead.
  dates = pd.date_range(
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=Tr
ue to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True
to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/humnkmhg.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmps0mtrkva/s1610954.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/s
tan_model/prophet_model.bin', 'random', 'seed=16952', 'data', 'file=/tmp/tmps0mtr
kva/humnkmhg.json', 'init=/tmp/tmps0mtrkva/s1610954.json', 'output', 'file=/tmp/t
mp003zrarn/prophet_modelqiuxplir/prophet_model-20250421111949.csv', 'method=optim
ize', 'algorithm=newton', 'iter=10000']
11:19:49 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:19:49 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarnin
g: 'Q' is deprecated and will be removed in a future version, please use 'QE' ins
tead.
  dates = pd.date_range(
```

In [86]:

```
model_global.plot(forecast_global)
```

Out[86]:





In [96]:

```
index_prop_res.to_excel('index_prop_res.xlsx')
index_prop_com.to_excel('index_prop_com.xlsx')
index_prop_vl.to_excel('index_prop_vl.xlsx')
index_prop_app.to_excel('index_prop_app.xlsx')
index_prop_in.to_excel('index_prop_in.xlsx')
index_prop_pu.to_excel('index_prop_pu.xlsx')
index_global.to_excel('index_global.xlsx')


beta_prop_res.to_excel('beta_prop_res.xlsx')
beta_prop_com.to_excel('beta_prop_com.xlsx')
beta_prop_vl.to_excel('beta_prop_vl.xlsx')
beta_prop_app.to_excel('beta_prop_app.xlsx')
beta_prop_in.to_excel('beta_prop_in.xlsx')
beta_prop_pu.to_excel('beta_prop_pu.xlsx')
beta_global.to_excel('beta_global.xlsx')



forecast_prop_res.to_excel('forecast_prop_res.xlsx')
forecast_prop_com.to_excel('forecast_prop_com.xlsx')
forecast_prop_vl.to_excel('forecast_prop_vl.xlsx')
forecast_prop_app.to_excel('forecast_prop_app.xlsx')
forecast_prop_in.to_excel('forecast_prop_in.xlsx')
forecast_prop_pu.to_excel('forecast_prop_pu.xlsx')
forecast_global.to_excel('forecast_global.xlsx')
```