phical-representation-ds-project-1

January 2, 2024

1 Data Visualization or Graphical Representation

[1]:
```
import matplotlib.pyplot as plt
```

[2]:
```
import numpy as np
import pandas as pd
```

2 Read data into Python

[5]:
```
project = pd.read_csv(r"/content/Datasets.csv")
```

3 Read data into Python

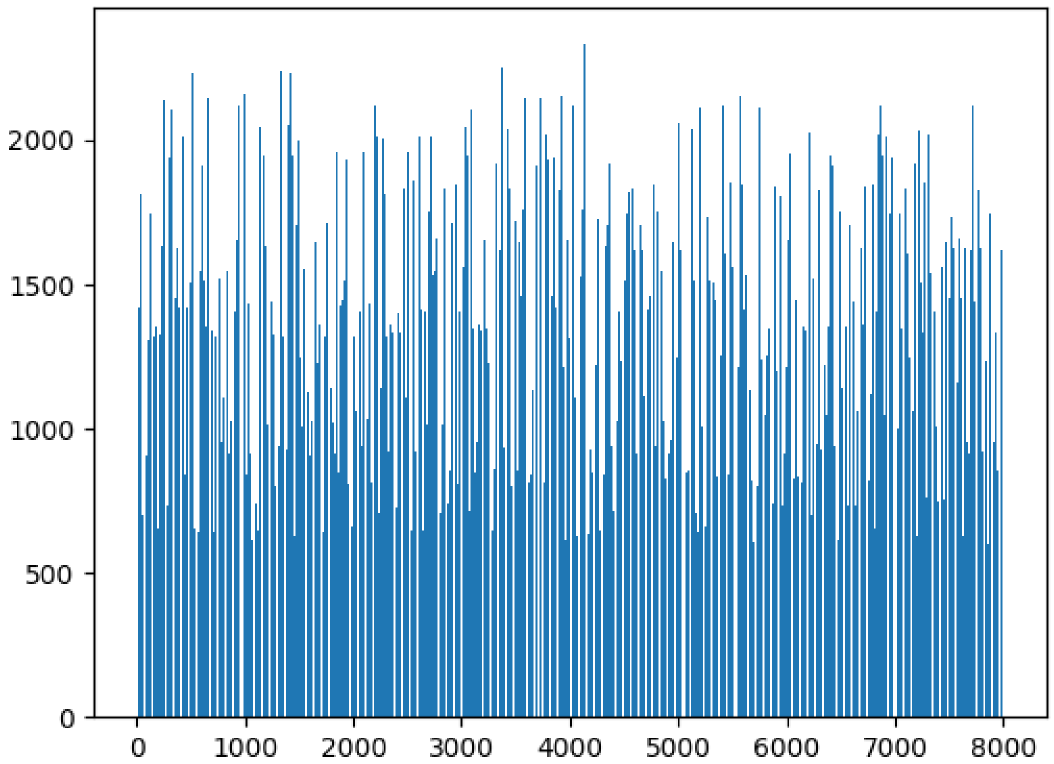
[6]:
```
project.shape
```

[6]:
```
(7999, 15)
```

4 barplot

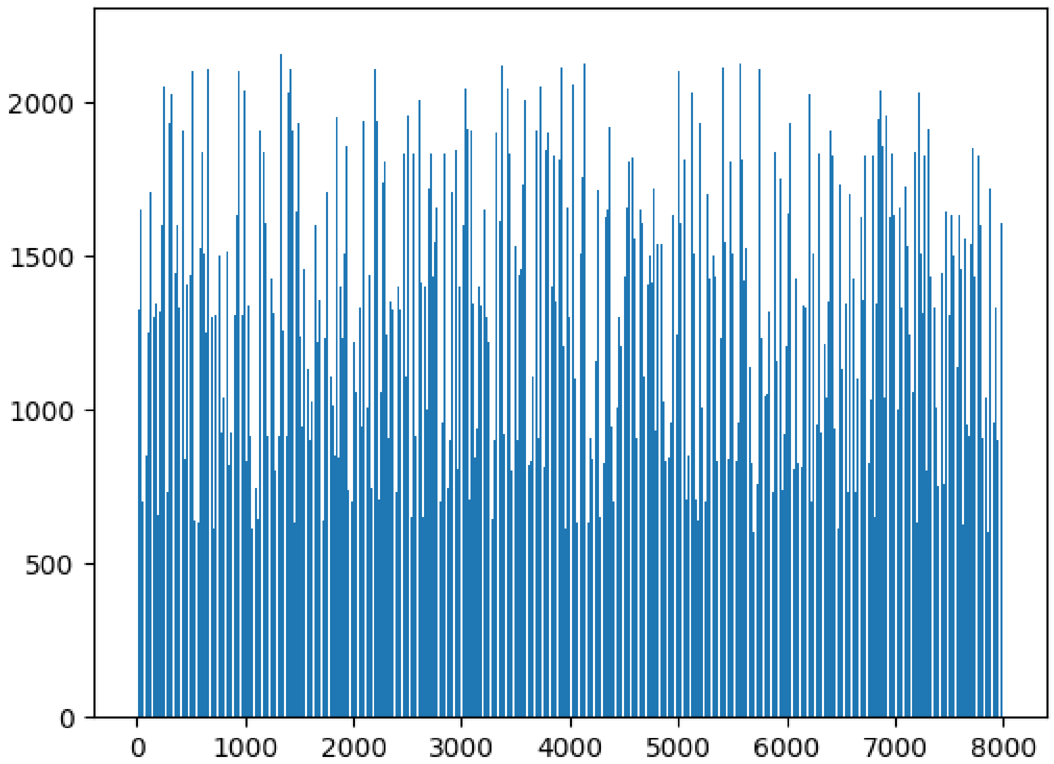
[7]:
```
plt.bar(height = project.Actual_Shipment_Time, x = np.arange(1, 8000, 1))
```

[7]:
```
<BarContainer object of 7999 artists>
```



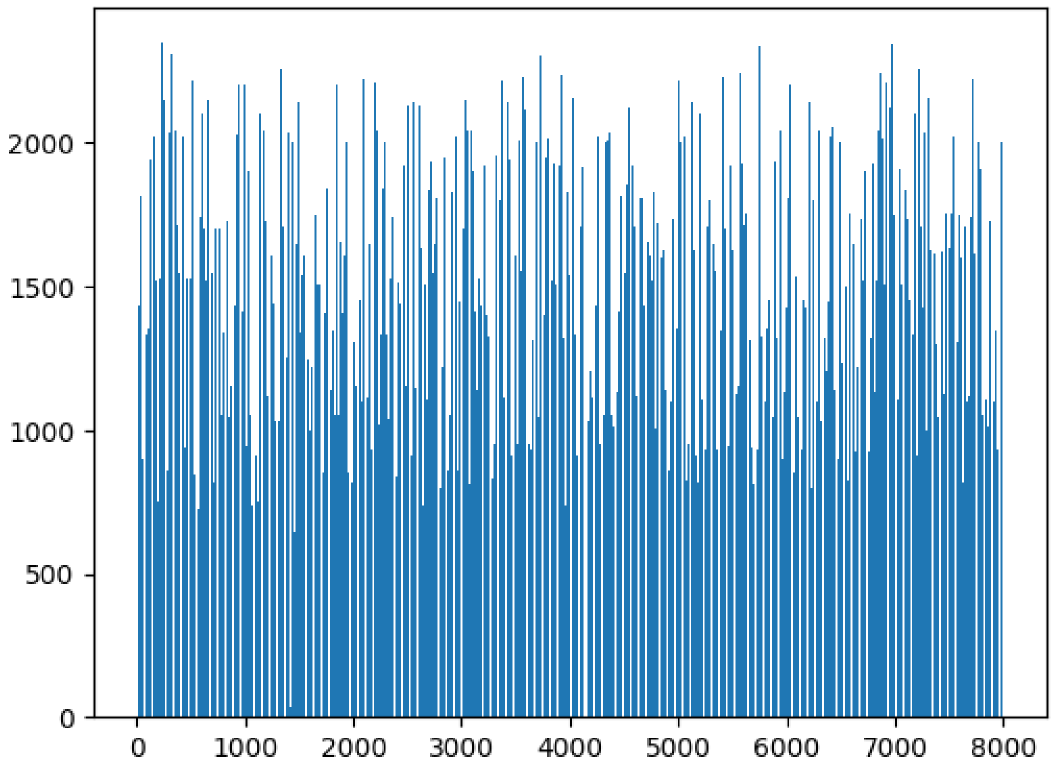
```
[8]: plt.bar(height = project.Planned_Shipment_Time, x = np.arange(1, 8000, 1))
```

```
[8]: <BarContainer object of 7999 artists>
```



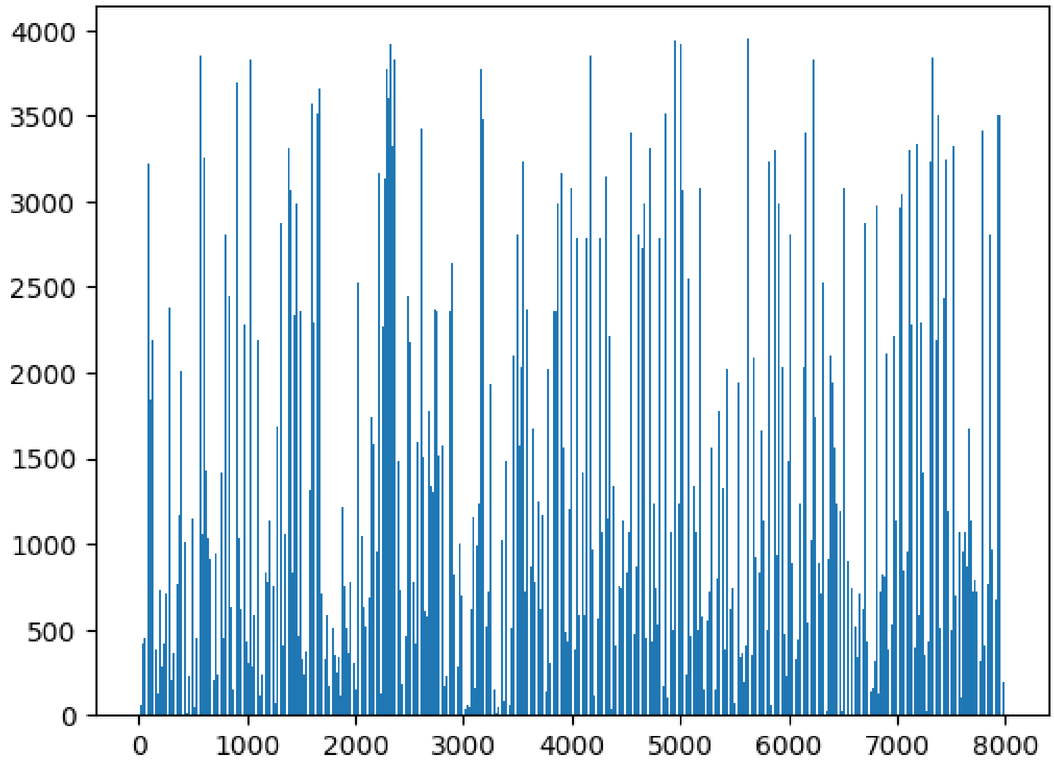
```
[9]: plt.bar(height = project.Planned_Delivery_Time, x = np.arange(1, 8000, 1))
```

```
[9]: <BarContainer object of 7999 artists>
```



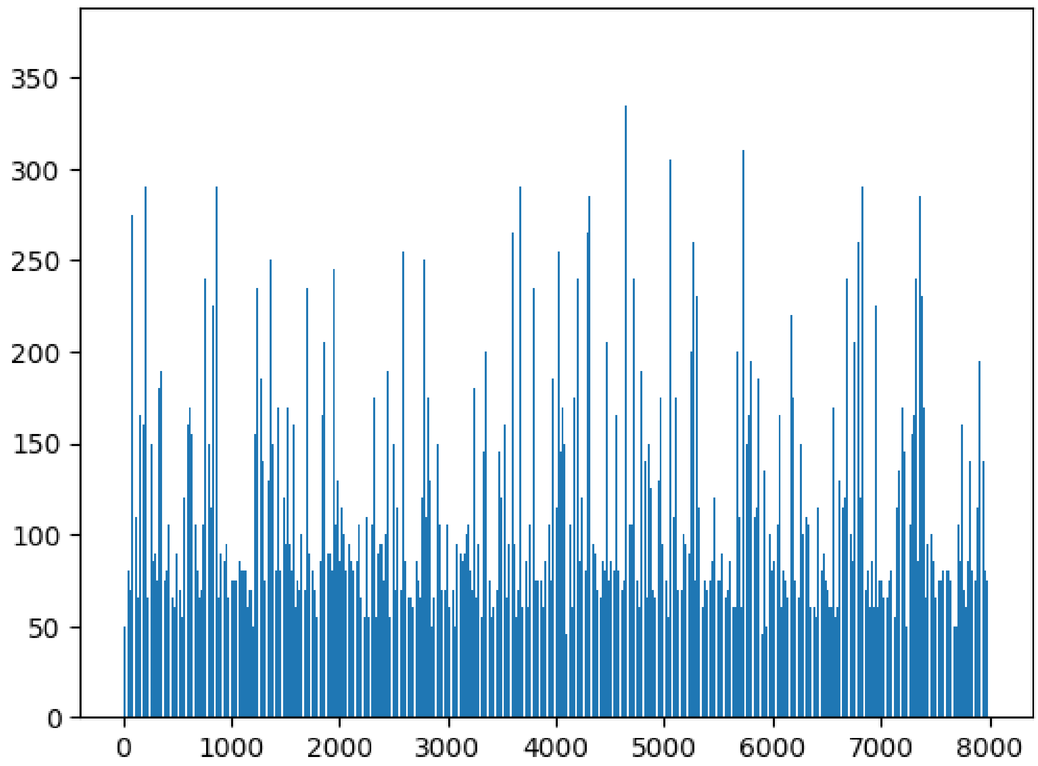
[10]:
```
plt.bar(height = project.Carrier_Num, x = np.arange(1, 8000, 1))
```

[10]:
```
<BarContainer object of 7999 artists>
```



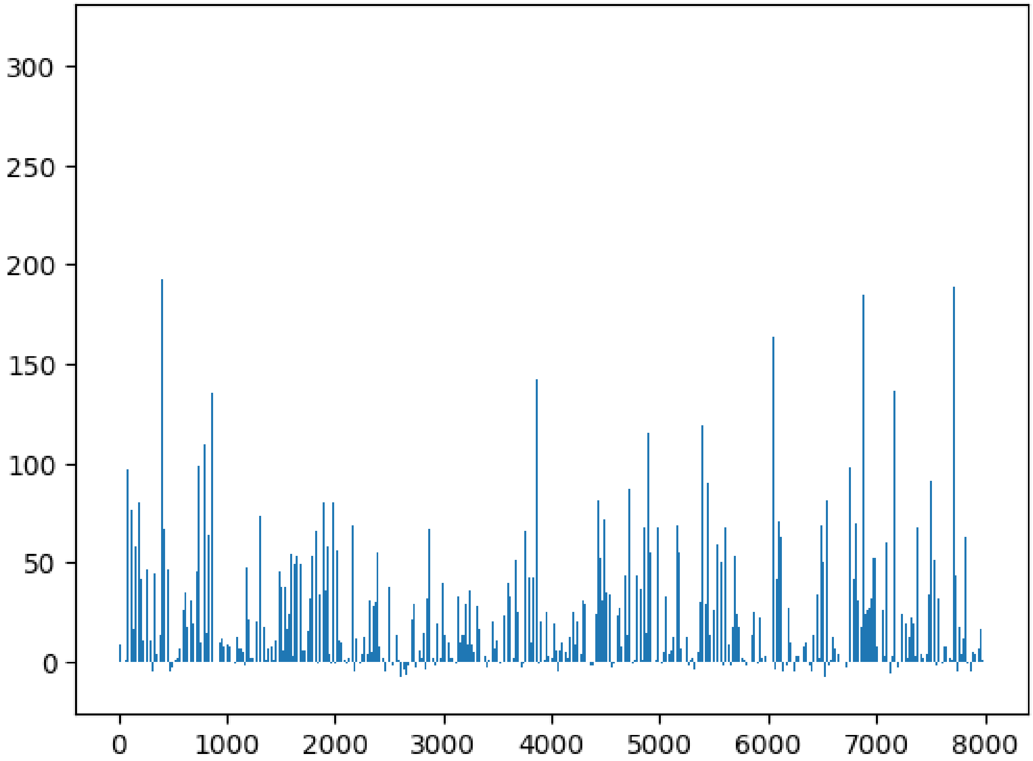
[11]:
```
plt.bar(height = project.Planned_TimeofTravel, x = np.arange(1, 8000, 1))
```

[11]:
```
<BarContainer object of 7999 artists>
```



[12]:
```
plt.bar(height = project.Shipment_Delay, x = np.arange(1, 8000, 1))
```

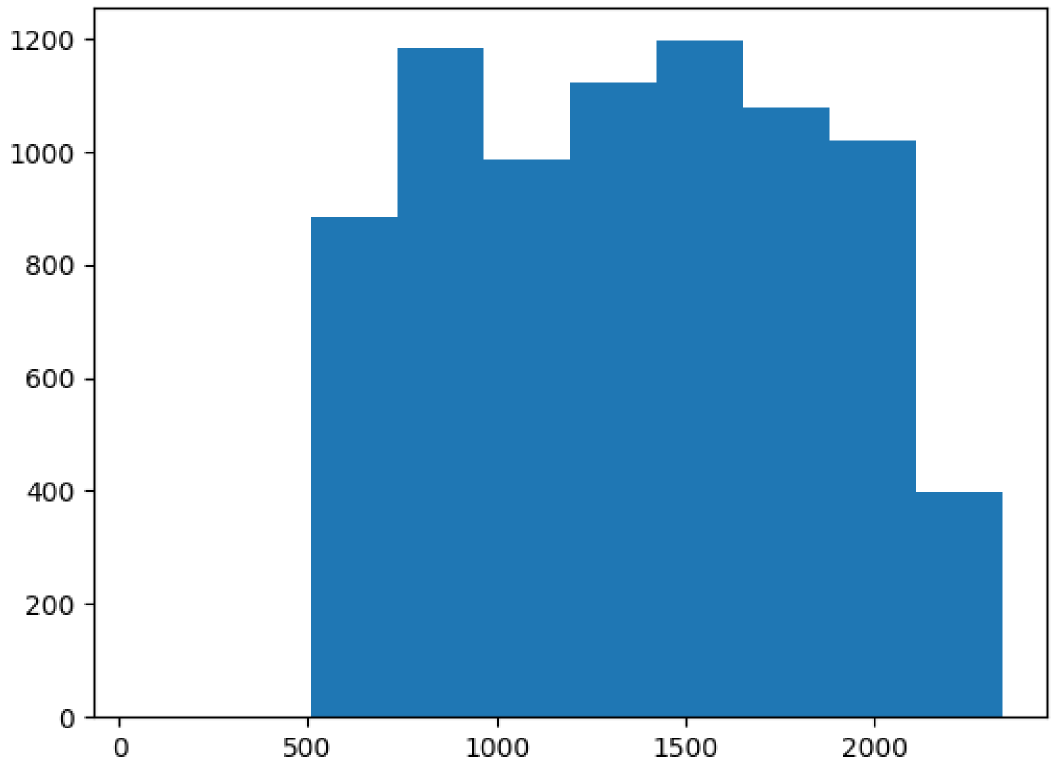
[12]:
```
<BarContainer object of 7999 artists>
```



5 Histogram

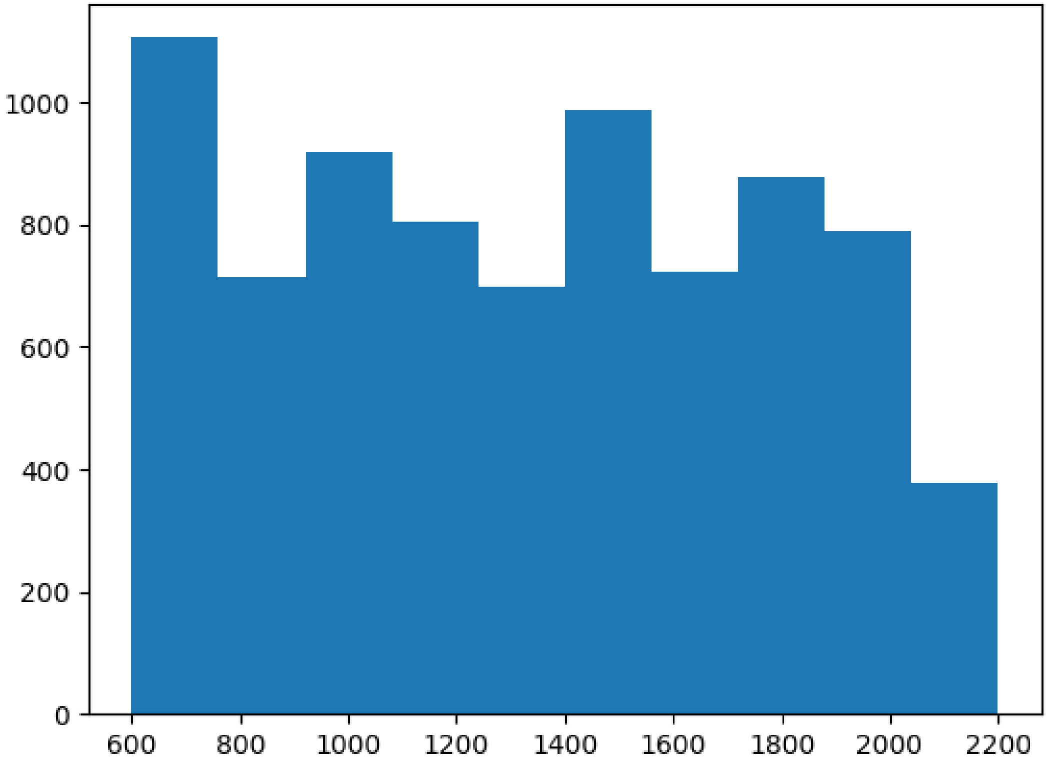
```
[14]: plt.hist(project.Actual_Shipment_Time)
```

```
[14]: (array([1.000e+00, 0.000e+00, 8.830e+02, 1.184e+03, 9.840e+02, 1.120e+03,
              1.195e+03, 1.076e+03, 1.020e+03, 3.970e+02]),
       array([  47. ,  276.4,  505.8,  735.2,  964.6, 1194. , 1423.4, 1652.8,
              1882.2, 2111.6, 2341. ]),
       <BarContainer object of 10 artists>)
```



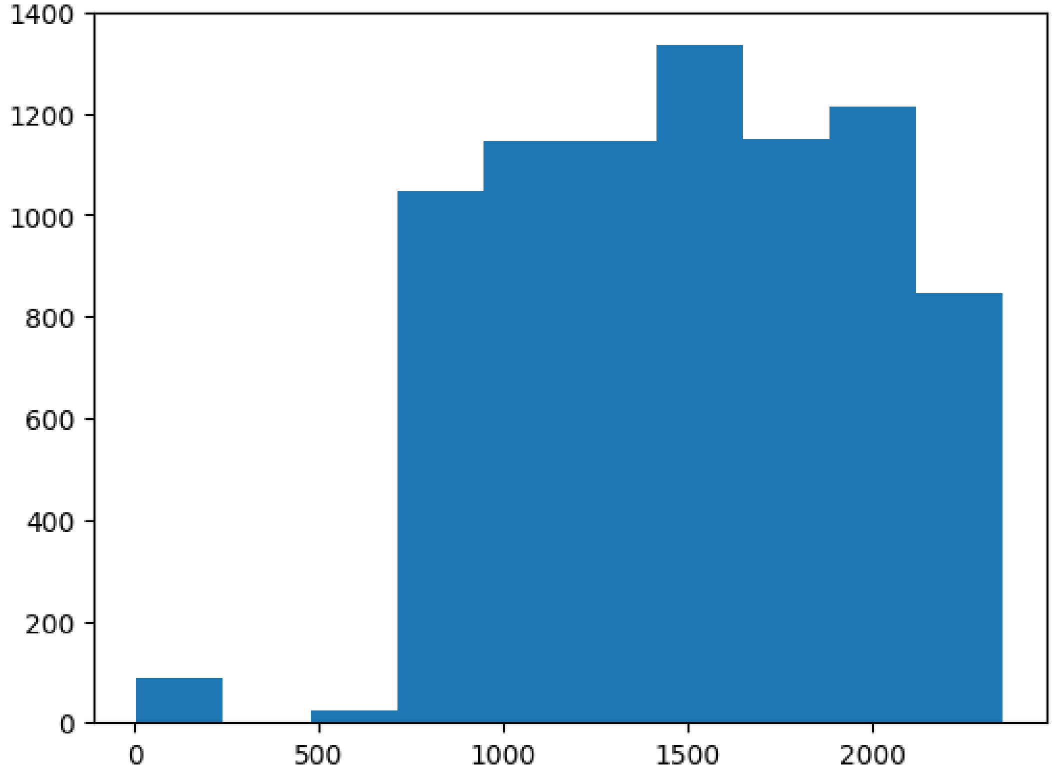
```
[15]: plt.hist(project.Planned_Shipment_Time)
```

```
[15]: (array([1106.,  714.,  917.,  805.,  699.,  988.,  725.,  878.,  788.,
                379.]),
       array([ 600.,  760.,  920., 1080., 1240., 1400., 1560., 1720., 1880.,
              2040., 2200.]),
       <BarContainer object of 10 artists>)
```



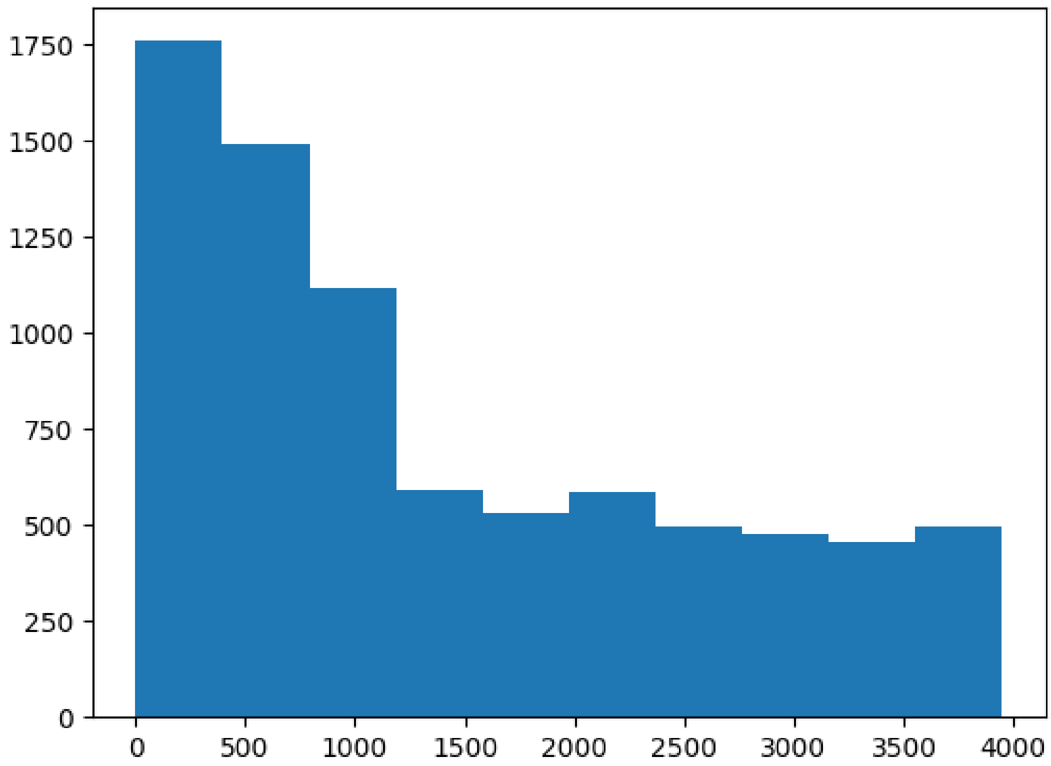
```
[16]: plt.hist(project.Planned_Delivery_Time)
```

```
[16]: (array([  89.,    0.,   25., 1047., 1147., 1146., 1334., 1149., 1215.,
               847.]),
       array([   5.,  240.,  475.,  710.,  945., 1180., 1415., 1650., 1885.,
              2120., 2355.]),
       <BarContainer object of 10 artists>)
```



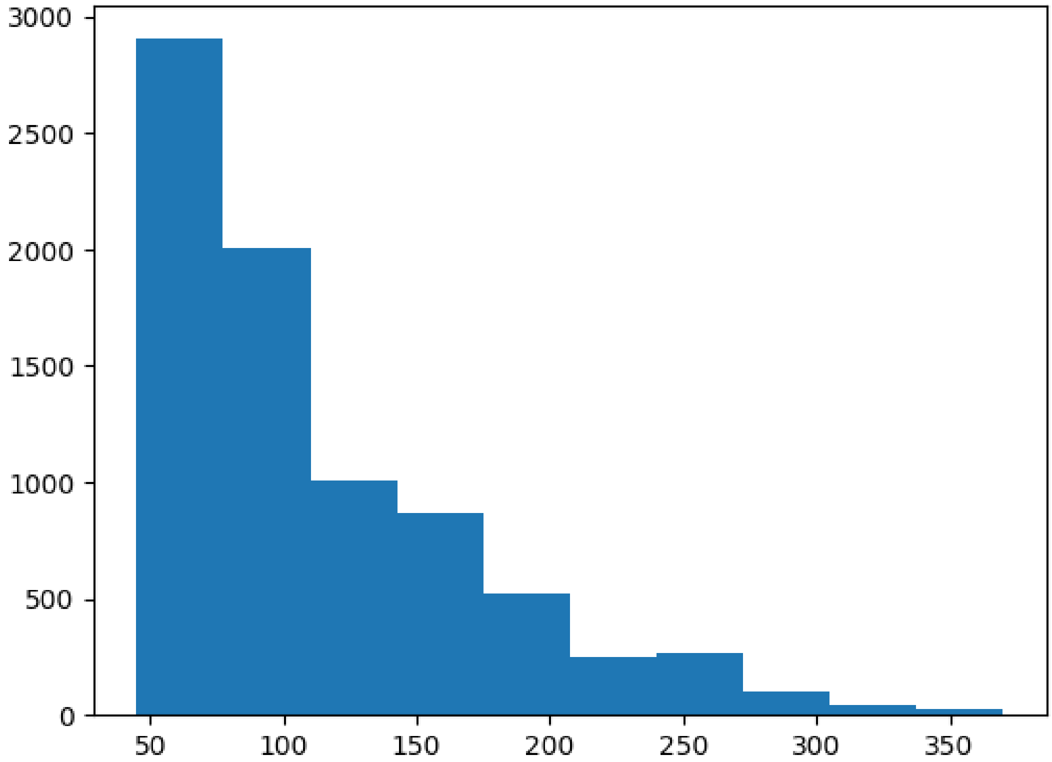
```
[17]: plt.hist(project.Carrier_Num)
```

```
[17]: (array([1758., 1490., 1115.,  593.,  531.,  588.,  497.,  476.,  455.,
         496.]),
 array([1.0000e+00, 3.9580e+02, 7.9060e+02, 1.1854e+03, 1.5802e+03,
        1.9750e+03, 2.3698e+03, 2.7646e+03, 3.1594e+03, 3.5542e+03,
        3.9490e+03]),
 <BarContainer object of 10 artists>)
```



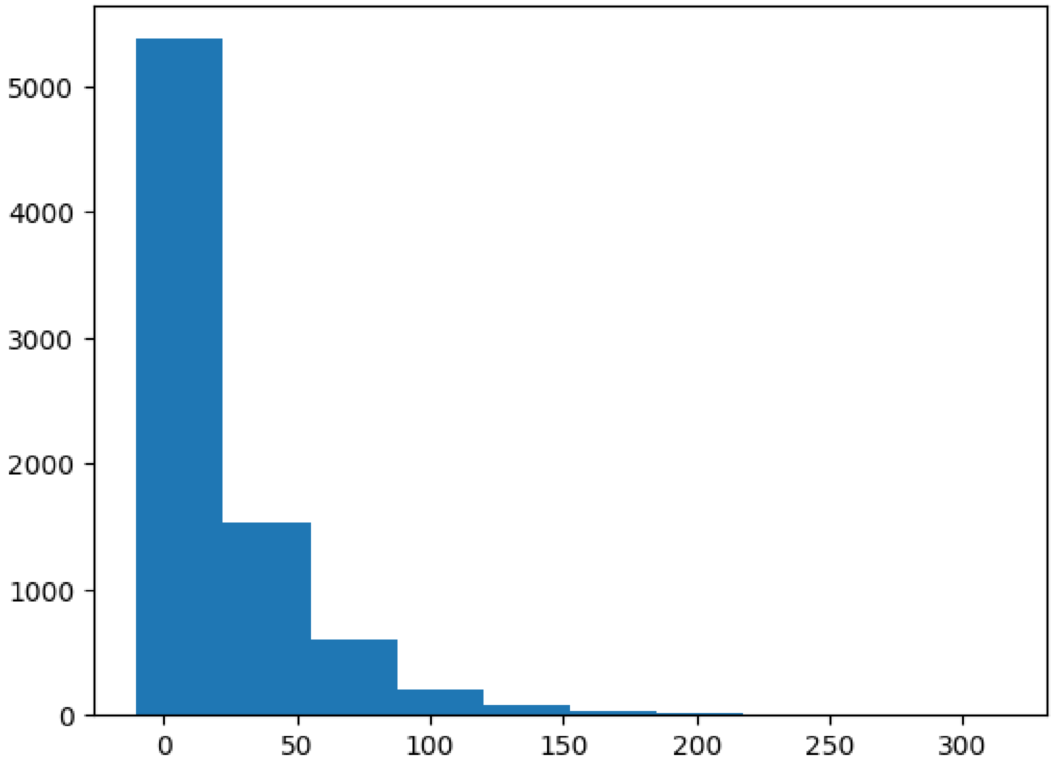
```
[18]: plt.hist(project.Planned_TimeofTravel)
```

```
[18]: (array([2904., 2007., 1009.,  869.,  520.,  252.,  269.,   98.,   41.,
                30.]),
       array([ 45. ,  77.5, 110. , 142.5, 175. , 207.5, 240. , 272.5, 305. ,
               337.5, 370. ]),
       <BarContainer object of 10 artists>)
```



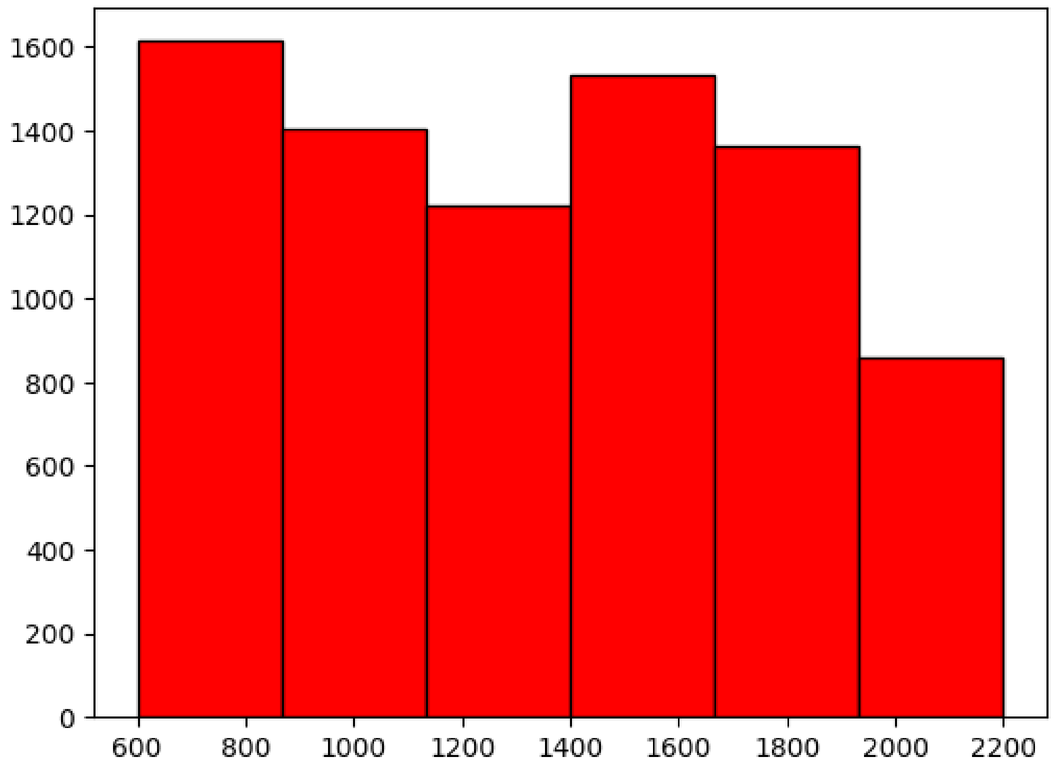
```
[19]: plt.hist(project.Shipment_Delay)
```

```
[19]: (array([5.373e+03, 1.532e+03, 5.950e+02, 2.020e+02, 7.900e+01, 4.000e+01,
              2.000e+01, 1.000e+01, 7.000e+00, 2.000e+00]),
       array([-10. ,  22.5,  55. ,  87.5, 120. , 152.5, 185. , 217.5, 250. ,
              282.5, 315. ]),
       <BarContainer object of 10 artists>)
```



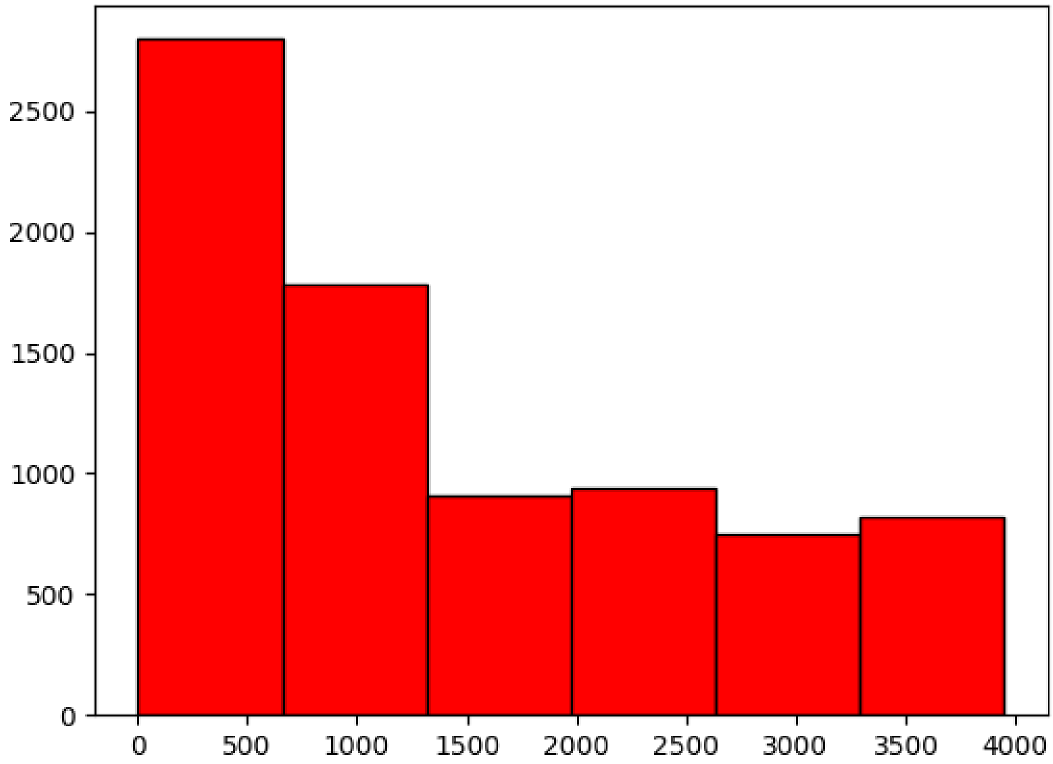
```
[20]: plt.hist(project.Planned_Shipment_Time, color='red', edgecolor = "black", bins␣
       ↪= 6)
```

```
[20]: (array([1614., 1406., 1221., 1533., 1364.,  861.]),
       array([ 600.        ,  866.66666667, 1133.33333333, 1400.        ,
              1666.66666667, 1933.33333333, 2200.        ]),
       <BarContainer object of 6 artists>)
```



```
[21]: plt.hist(project.Carrier_Num, color='red', edgecolor = "black", bins = 6)
```

```
[21]: (array([2797., 1783.,  907.,  940.,  748.,  824.]),
       array([1.000e+00, 6.590e+02, 1.317e+03, 1.975e+03, 2.633e+03, 3.291e+03,
              3.949e+03]),
       <BarContainer object of 6 artists>)
```



[22]:
```
help(plt.hist)
```

```
Help on function hist in module matplotlib.pyplot:

hist(x, bins=None, range=None, density=False, weights=None, cumulative=False,
bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None,
log=False, color=None, label=None, stacked=False, *, data=None, **kwargs)
    Compute and plot a histogram.

    This method uses `numpy.histogram` to bin the data in *x* and count the
    number of values in each bin, then draws the distribution either as a
    `.BarContainer` or `.Polygon`. The *bins*, *range*, *density*, and
    *weights* parameters are forwarded to `numpy.histogram`.

    If the data has already been binned and counted, use `~.bar` or
    `~.stairs` to plot the distribution::

        counts, bins = np.histogram(x)
        plt.stairs(counts, bins)

    Alternatively, plot pre-computed bins and counts using ``hist()`` by
    treating each bin as a single point with a weight equal to its count::
```

```
    plt.hist(bins[:-1], bins, weights=counts)

The data input *x* can be a singular array, a list of datasets of
potentially different lengths ([*x0*, *x1*, …]), or a 2D ndarray in
which each column is a dataset. Note that the ndarray form is
transposed relative to the list form. If the input is an array, then
the return value is a tuple (*n*, *bins*, *patches*); if the input is a
sequence of arrays, then the return value is a tuple
([*n0*, *n1*, …], *bins*, [*patches0*, *patches1*, …]).

Masked arrays are not supported.

Parameters
----------
x : (n,) array or sequence of (n,) arrays
    Input values, this takes either a single array or a sequence of
    arrays which are not required to be of the same length.

bins : int or sequence or str, default: :rc:`hist.bins`
    If *bins* is an integer, it defines the number of equal-width bins
    in the range.

    If *bins* is a sequence, it defines the bin edges, including the
    left edge of the first bin and the right edge of the last bin;
    in this case, bins may be unequally spaced.  All but the last
    (righthand-most) bin is half-open.  In other words, if *bins* is::

        [1, 2, 3, 4]

    then the first bin is ``[1, 2)`` (including 1, but excluding 2) and
    the second ``[2, 3)``.  The last bin, however, is ``[3, 4]``, which
    *includes* 4.

    If *bins* is a string, it is one of the binning strategies
    supported by `numpy.histogram_bin_edges`: 'auto', 'fd', 'doane',
    'scott', 'stone', 'rice', 'sturges', or 'sqrt'.

range : tuple or None, default: None
    The lower and upper range of the bins. Lower and upper outliers
    are ignored. If not provided, *range* is ``(x.min(), x.max())``.
    Range has no effect if *bins* is a sequence.

    If *bins* is a sequence or *range* is specified, autoscaling
    is based on the specified bin range instead of the
    range of x.

density : bool, default: False
```

```
    If ``True``, draw and return a probability density: each bin
    will display the bin's raw count divided by the total number of
    counts *and the bin width*
    (``density = counts / (sum(counts) * np.diff(bins))``),
    so that the area under the histogram integrates to 1
    (``np.sum(density * np.diff(bins)) == 1``).

    If *stacked* is also ``True``, the sum of the histograms is
    normalized to 1.

weights : (n,) array-like or None, default: None
    An array of weights, of the same shape as *x*.  Each value in
    *x* only contributes its associated weight towards the bin count
    (instead of 1).  If *density* is ``True``, the weights are
    normalized, so that the integral of the density over the range
    remains 1.

cumulative : bool or -1, default: False
    If ``True``, then a histogram is computed where each bin gives the
    counts in that bin plus all bins for smaller values. The last bin
    gives the total number of datapoints.

    If *density* is also ``True`` then the histogram is normalized such
    that the last bin equals 1.

    If *cumulative* is a number less than 0 (e.g., -1), the direction
    of accumulation is reversed.  In this case, if *density* is also
    ``True``, then the histogram is normalized such that the first bin
    equals 1.

bottom : array-like, scalar, or None, default: None
    Location of the bottom of each bin, i.e. bins are drawn from
    ``bottom`` to ``bottom + hist(x, bins)`` If a scalar, the bottom
    of each bin is shifted by the same amount. If an array, each bin
    is shifted independently and the length of bottom must match the
    number of bins. If None, defaults to 0.

histtype : {'bar', 'barstacked', 'step', 'stepfilled'}, default: 'bar'
    The type of histogram to draw.

    - 'bar' is a traditional bar-type histogram.  If multiple data
      are given the bars are arranged side by side.
    - 'barstacked' is a bar-type histogram where multiple
      data are stacked on top of each other.
    - 'step' generates a lineplot that is by default unfilled.
    - 'stepfilled' generates a lineplot that is by default filled.

align : {'left', 'mid', 'right'}, default: 'mid'
```

```
    The horizontal alignment of the histogram bars.

    - 'left': bars are centered on the left bin edges.
    - 'mid': bars are centered between the bin edges.
    - 'right': bars are centered on the right bin edges.

orientation : {'vertical', 'horizontal'}, default: 'vertical'
    If 'horizontal', `~.Axes.barh` will be used for bar-type histograms
    and the *bottom* kwarg will be the left edges.

rwidth : float or None, default: None
    The relative width of the bars as a fraction of the bin width.  If
    ``None``, automatically compute the width.

    Ignored if *histtype* is 'step' or 'stepfilled'.

log : bool, default: False
    If ``True``, the histogram axis will be set to a log scale.

color : color or array-like of colors or None, default: None
    Color or sequence of colors, one per dataset.  Default (``None``)
    uses the standard line color sequence.

label : str or None, default: None
    String, or sequence of strings to match multiple datasets.  Bar
    charts yield multiple patches per dataset, but only the first gets
    the label, so that `~.Axes.legend` will work as expected.

stacked : bool, default: False
    If ``True``, multiple data are stacked on top of each other If
    ``False`` multiple data are arranged side by side if histtype is
    'bar' or on top of each other if histtype is 'step'

Returns
-------
n : array or list of arrays
    The values of the histogram bins. See *density* and *weights* for a
    description of the possible semantics.  If input *x* is an array,
    then this is an array of length *nbins*. If input is a sequence of
    arrays ``[data1, data2, …]``, then this is a list of arrays with
    the values of the histograms for each of the arrays in the same
    order.  The dtype of the array *n* (or of its element arrays) will
    always be float even if no weighting or normalization is used.

bins : array
    The edges of the bins. Length nbins + 1 (nbins left edges and right
    edge of last bin).  Always a single array even when multiple data
    sets are passed in.
    patches : `.BarContainer` or list of a single `.Polygon` or list of such
objects
        Container of individual artists used to create the histogram
        or list of such containers if there are multiple input datasets.

    Other Parameters
    ----------------
    data : indexable object, optional
        If given, the following parameters also accept a string ``s``, which is
        interpreted as ``data[s]`` (unless this raises an exception):

        *x*, *weights*

    **kwargs
        `~matplotlib.patches.Patch` properties

    See Also
    --------
    hist2d : 2D histogram with rectangular bins
    hexbin : 2D histogram with hexagonal bins

    Notes
    -----
    For large numbers of bins (>1000), plotting can be significantly faster
    if *histtype* is set to 'step' or 'stepfilled' rather than 'bar' or
    'barstacked'.
```

6 Histogram using Seaborn

[23]:

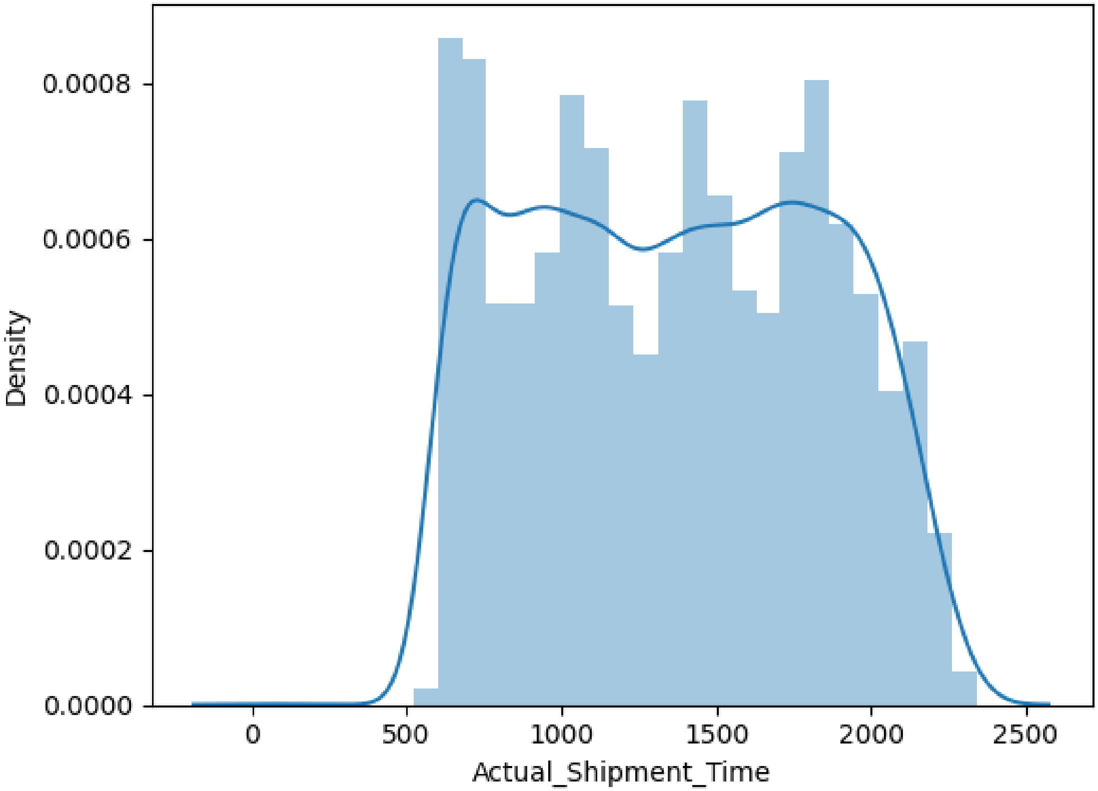
```
import seaborn as sns
```

[24]:

```
sns.distplot(project.Actual_Shipment_Time)
```

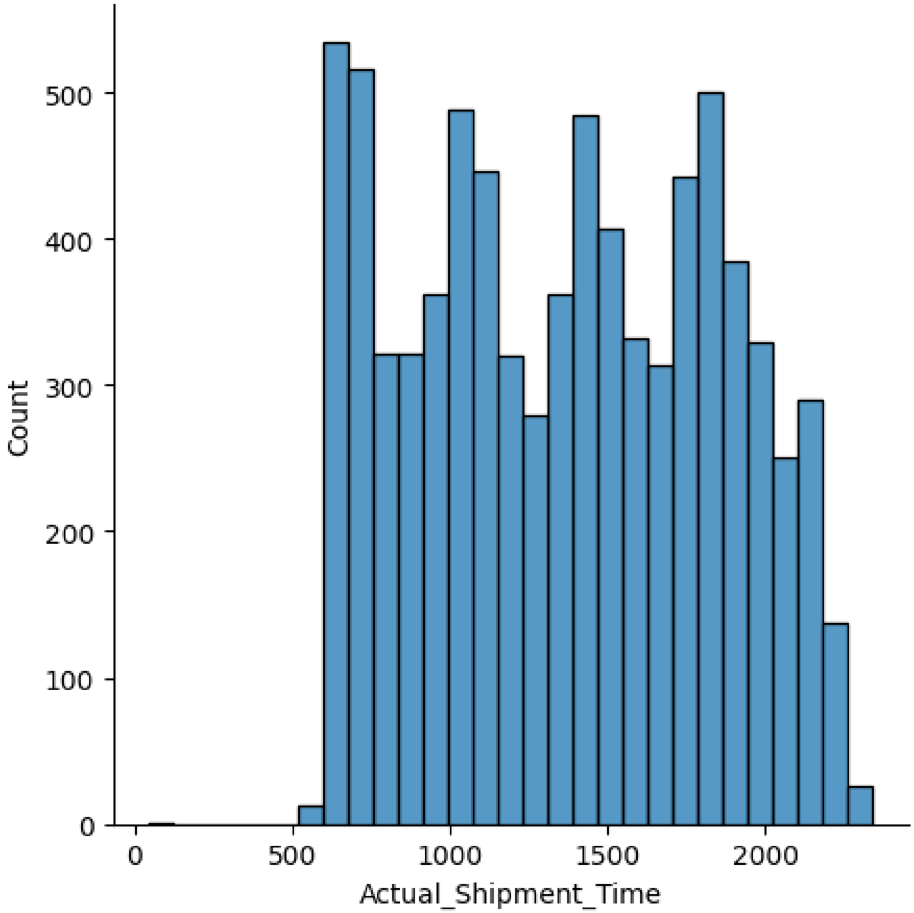
```
<ipython-input-24-67120540fb27>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(project.Actual_Shipment_Time)
```

```
[24]: <Axes: xlabel='Actual_Shipment_Time', ylabel='Density'>
```



```
[25]: sns.displot(project.Actual_Shipment_Time)
```

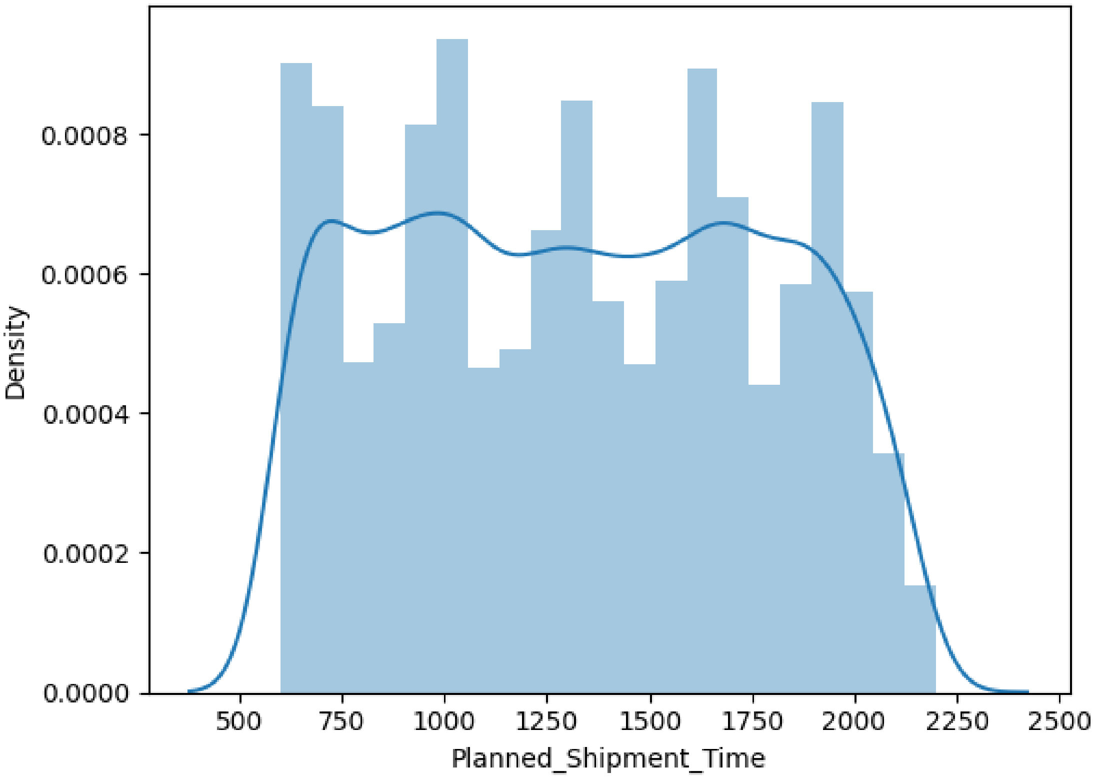
```
[25]: <seaborn.axisgrid.FacetGrid at 0x7fe0564effd0>
```



[26]:
```
sns.distplot(project.Planned_Shipment_Time)
```

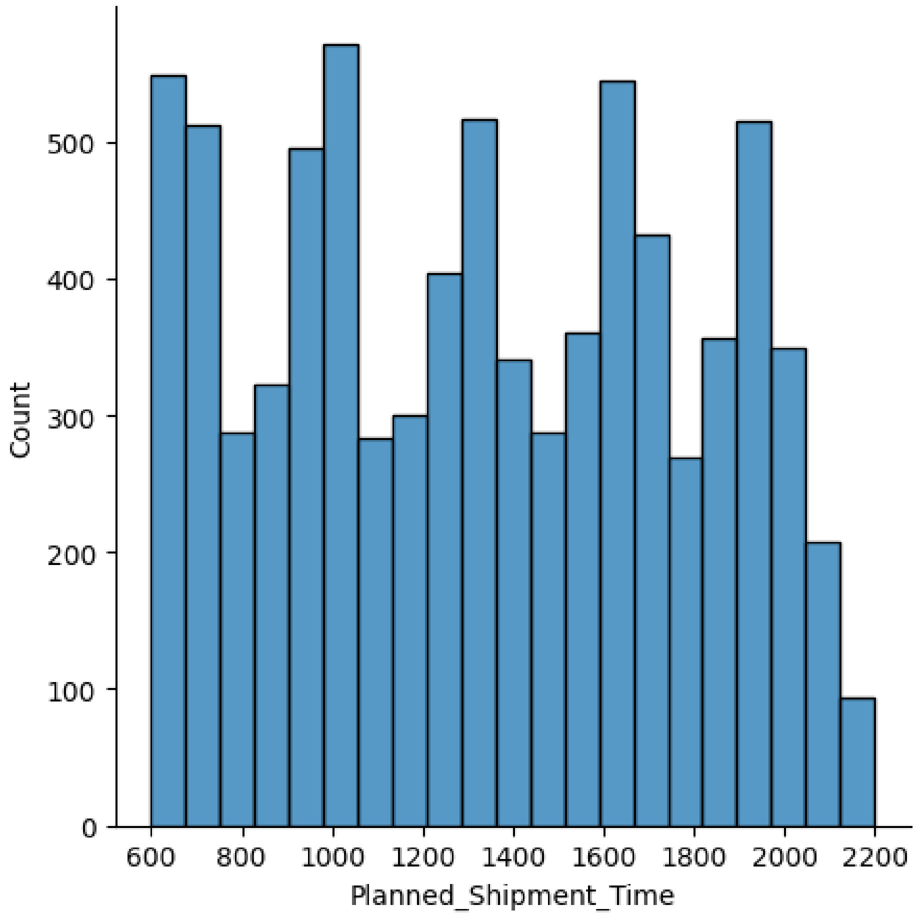
```
<ipython-input-26-7a7907fb2066>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(project.Planned_Shipment_Time)
```

[26]:
```
<Axes: xlabel='Planned_Shipment_Time', ylabel='Density'>
```



[27]:
```
sns.displot(project.Planned_Shipment_Time)
```

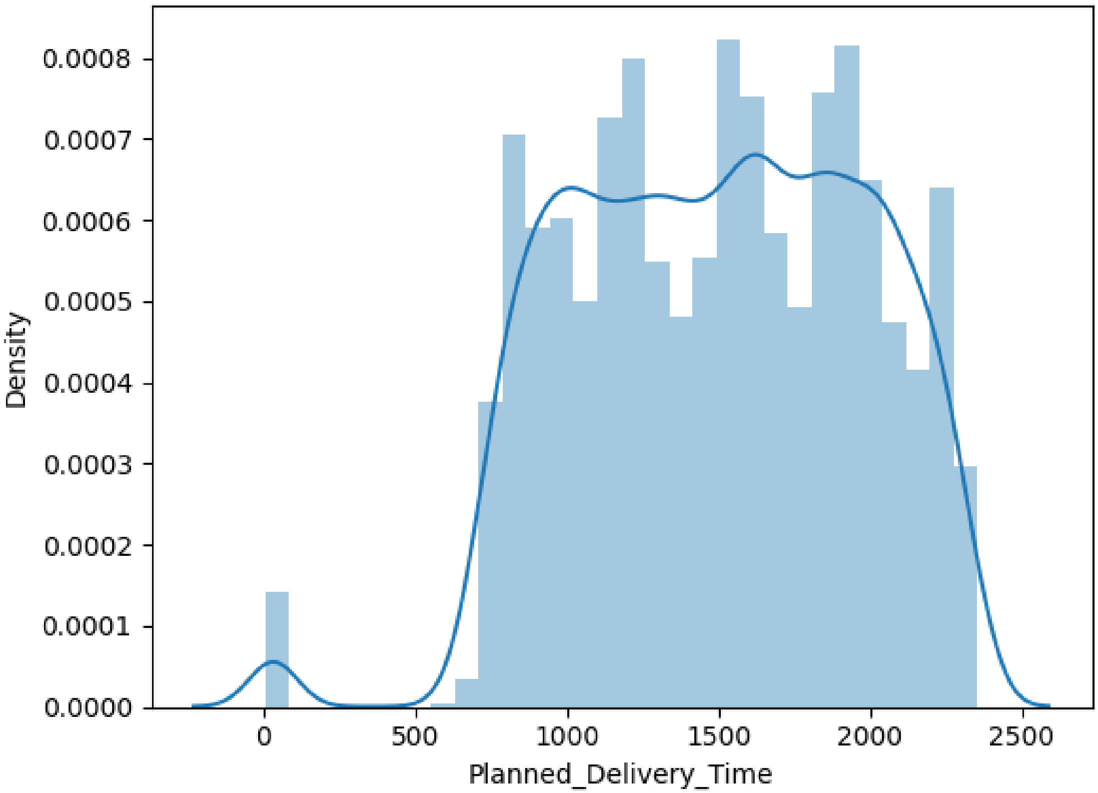
[27]:
```
<seaborn.axisgrid.FacetGrid at 0x7fe0543efc40>
```



[28]:
```
sns.distplot(project.Planned_Delivery_Time)
```

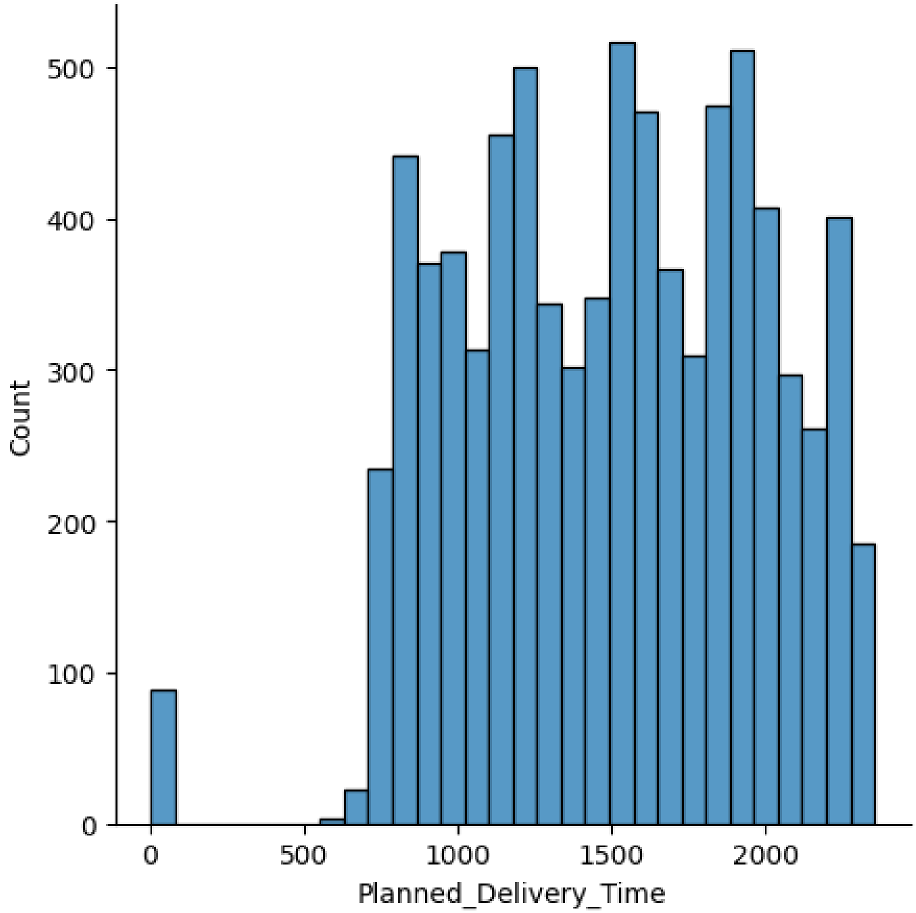
```
<ipython-input-28-4422adac3f3c>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(project.Planned_Delivery_Time)
```

[28]:
```
<Axes: xlabel='Planned_Delivery_Time', ylabel='Density'>
```



[29]:
```
sns.displot(project.Planned_Delivery_Time)
```

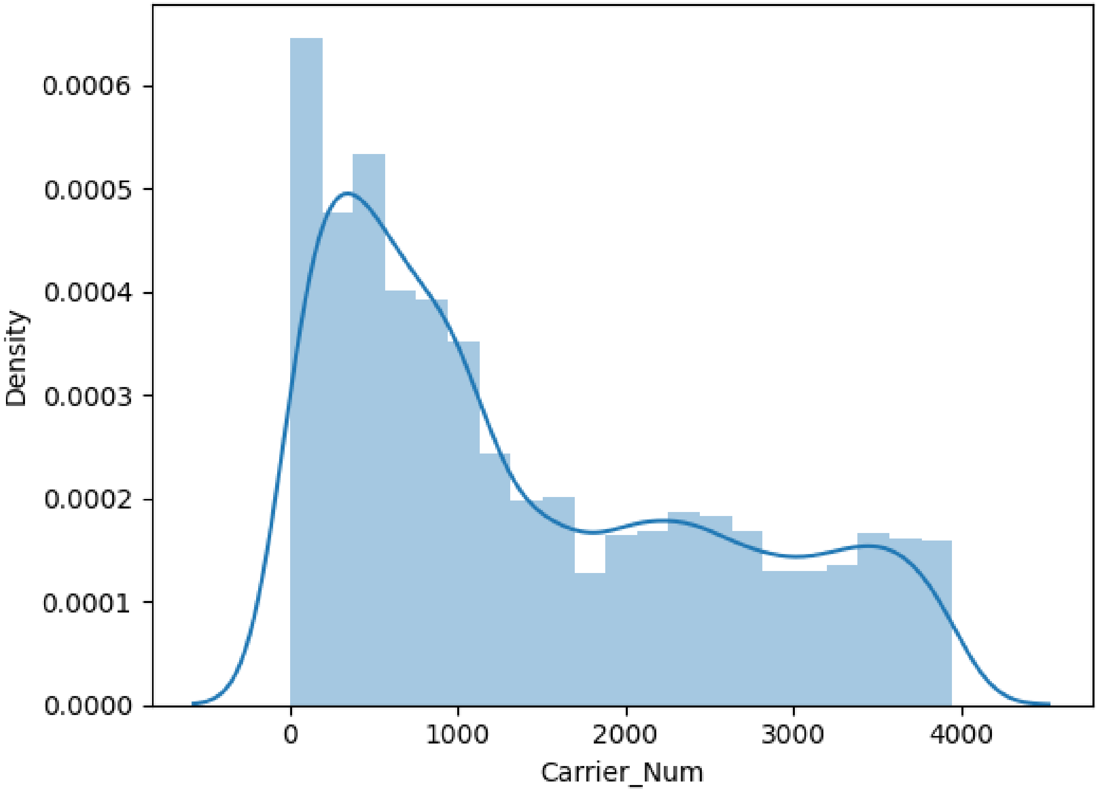
[29]:
```
<seaborn.axisgrid.FacetGrid at 0x7fe0541f1c60>
```



[30]:
```
sns.distplot(project.Carrier_Num)
```

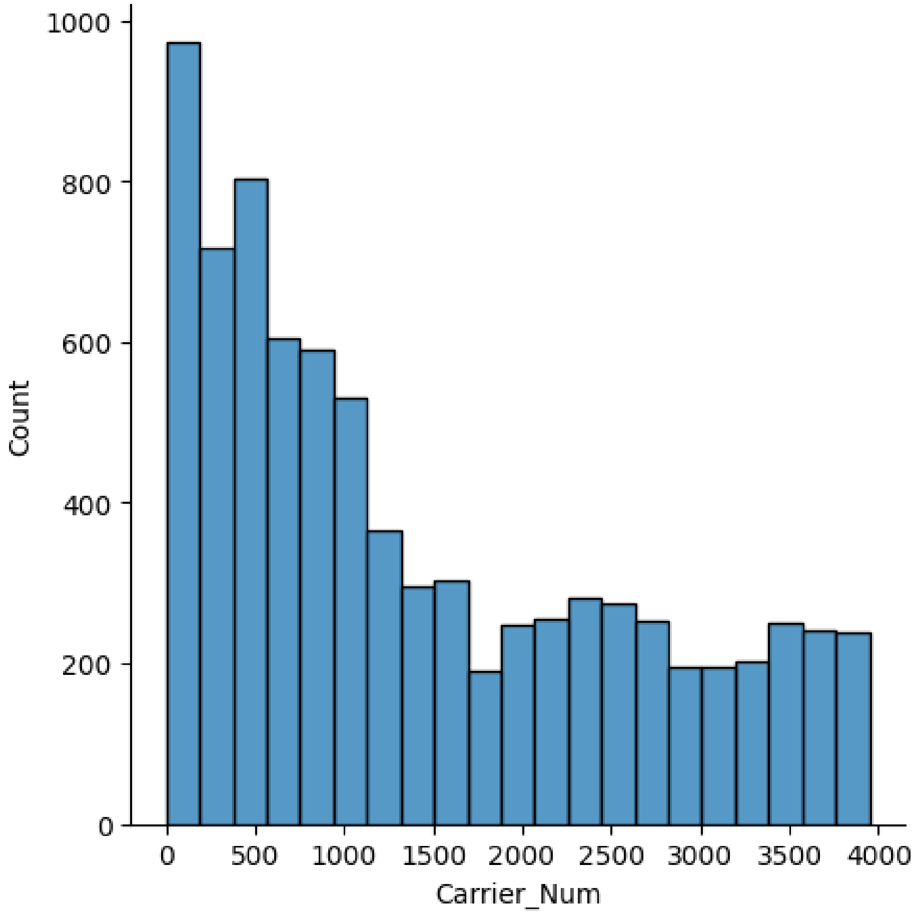
```
<ipython-input-30-1ac42355d1d7>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(project.Carrier_Num)
```

[30]:
```
<Axes: xlabel='Carrier_Num', ylabel='Density'>
```



```
[31]: sns.displot(project.Carrier_Num)
```

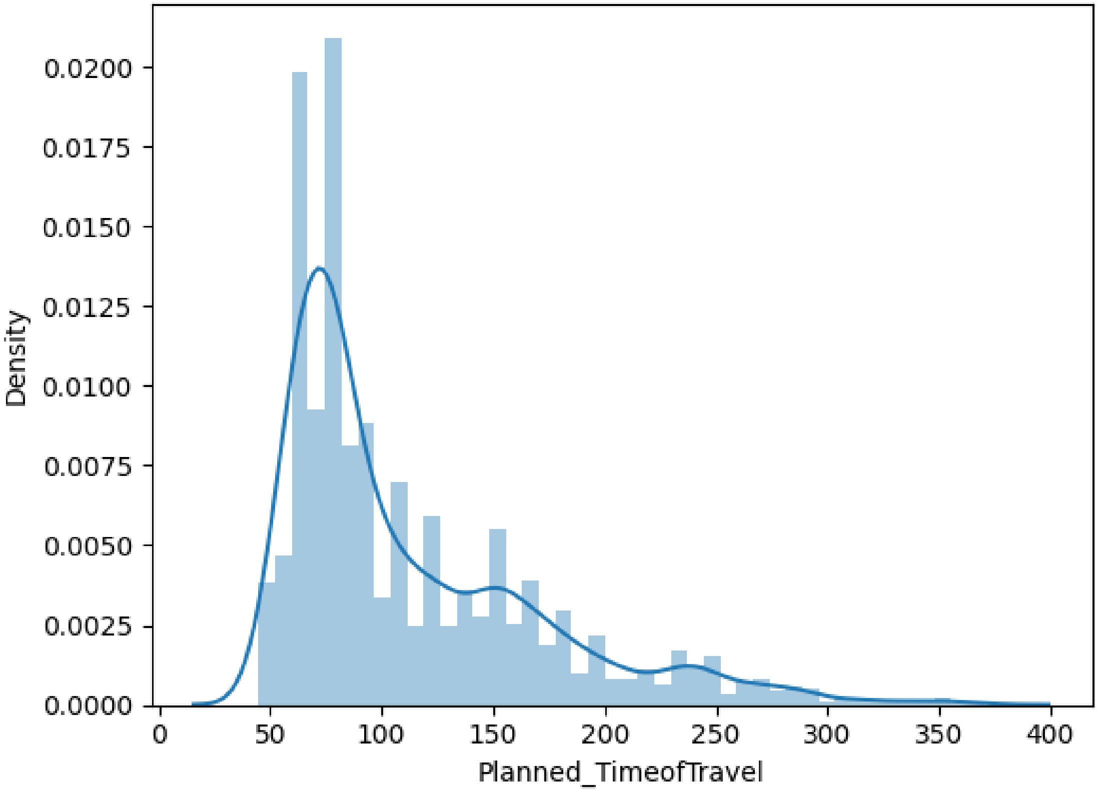
```
[31]: <seaborn.axisgrid.FacetGrid at 0x7fe0541b7760>
```



```
[32]: sns.distplot(project.Planned_TimeofTravel)
```

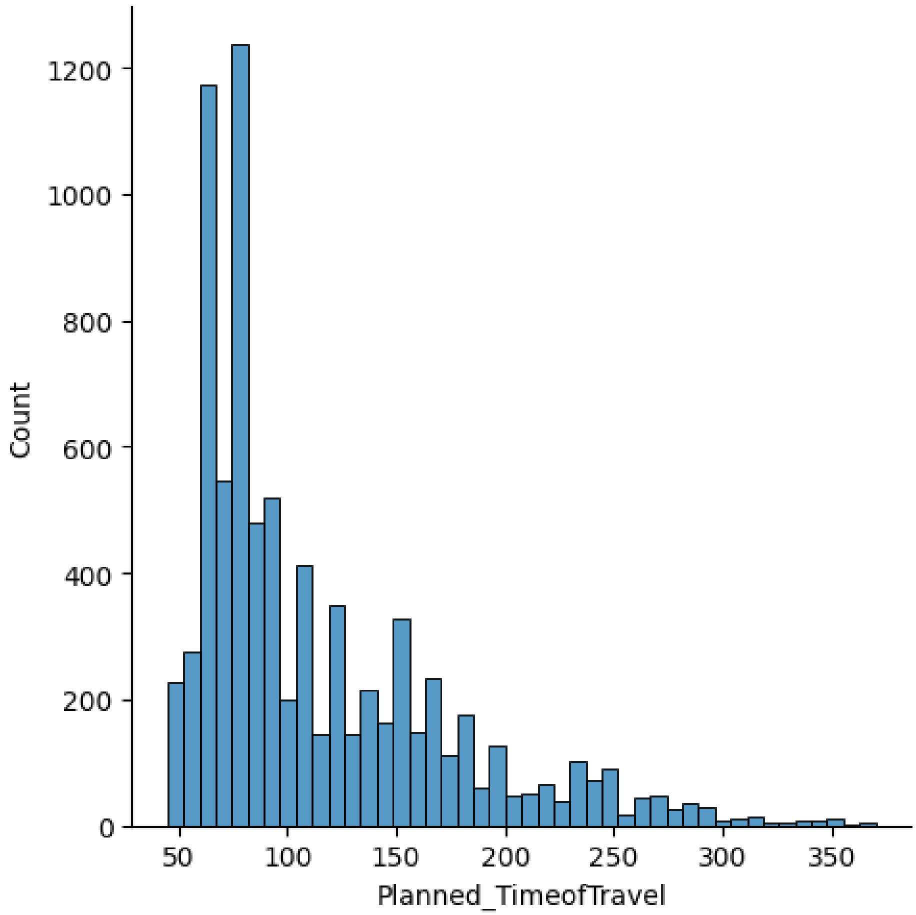
```
<ipython-input-32-d8bb81c4b701>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(project.Planned_TimeofTravel)
```

```
[32]: <Axes: xlabel='Planned_TimeofTravel', ylabel='Density'>
```



```
[33]: sns.displot(project.Planned_TimeofTravel)
```

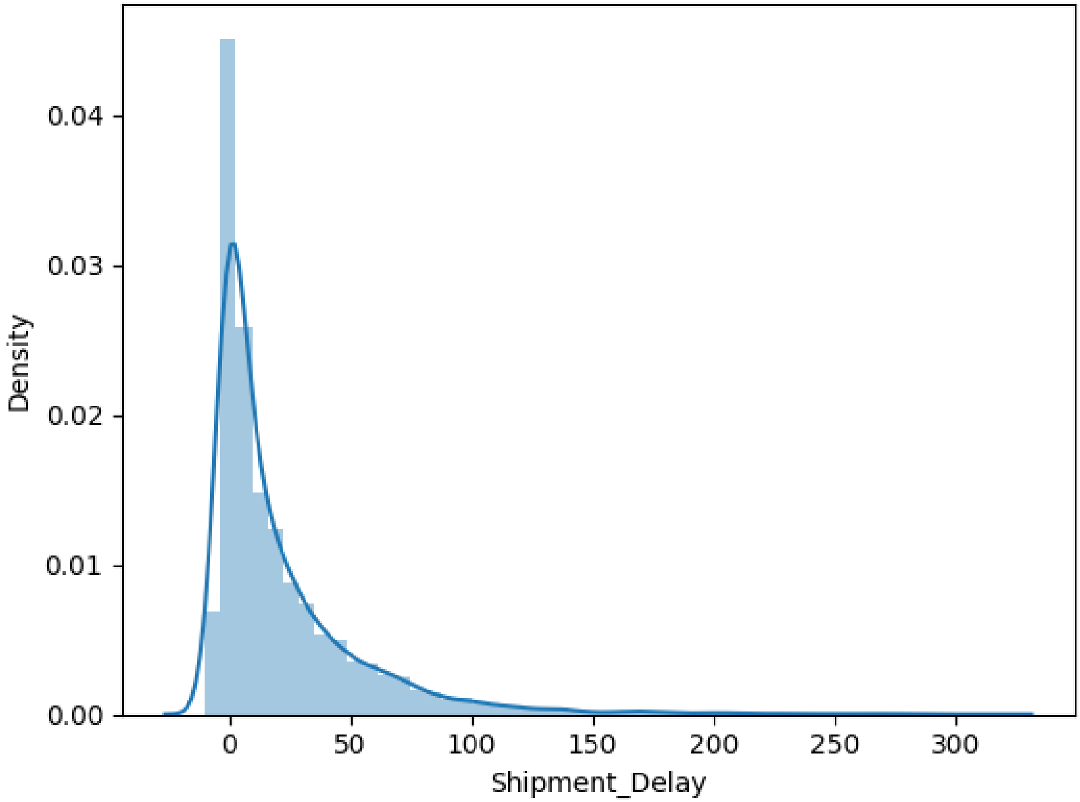
```
[33]: <seaborn.axisgrid.FacetGrid at 0x7fe0567cff10>
```



```
[34]: sns.distplot(project.Shipment_Delay)
```

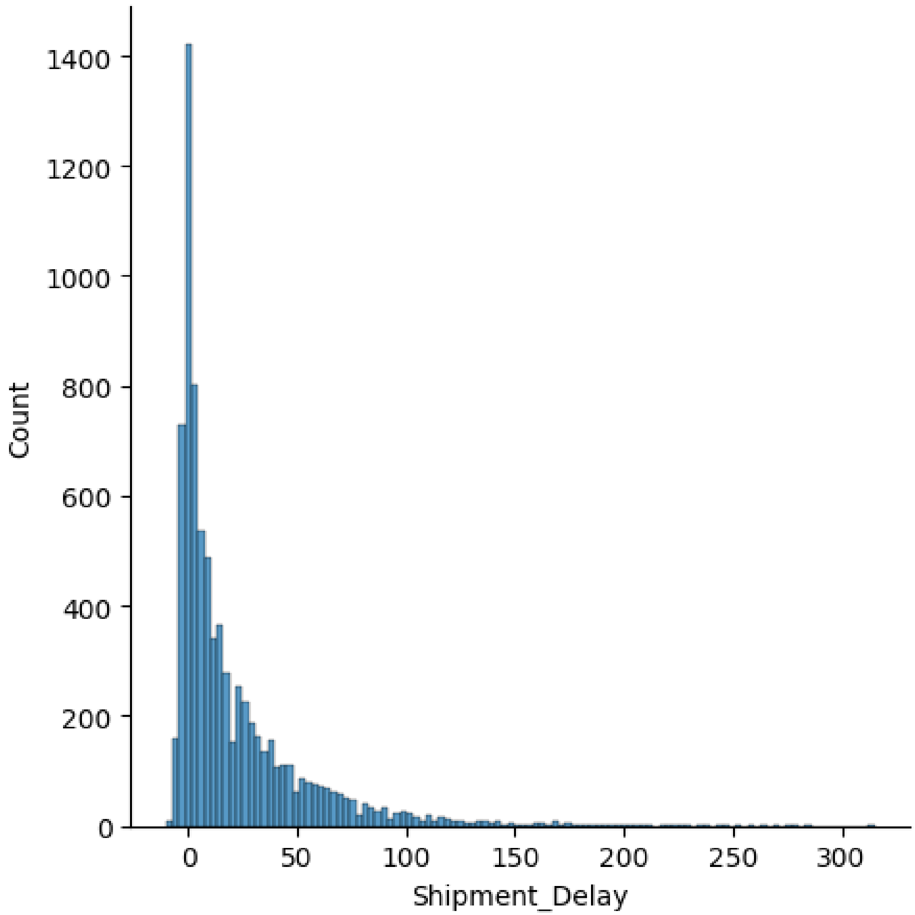
```
<ipython-input-34-144c4f121fc1>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(project.Shipment_Delay)
```

```
[34]: <Axes: xlabel='Shipment_Delay', ylabel='Density'>
```



```
[35]: sns.displot(project.Shipment_Delay)
```

```
[35]: <seaborn.axisgrid.FacetGrid at 0x7fe053ce6860>
```



7 Boxplot

[36]:
```
plt.figure()
```

[36]:
```
<Figure size 640x480 with 0 Axes>
```

```
<Figure size 640x480 with 0 Axes>
```

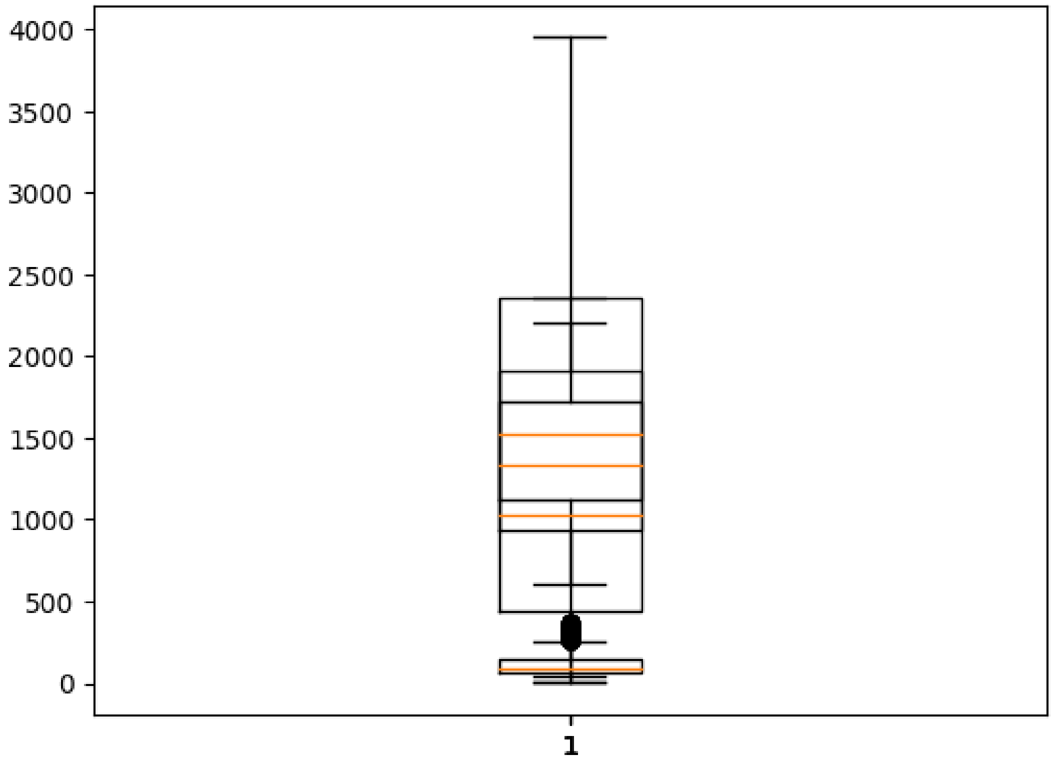
[37]:
```
plt.boxplot(project.Actual_Shipment_Time)
```

```
Help on function boxplot in module matplotlib.pyplot:

boxplot(x, notch=None, sym=None, vert=None, whis=None, positions=None,
widths=None, patch_artist=None, bootstrap=None, usermedians=None,
conf_intervals=None, meanline=None, showmeans=None, showcaps=None, showbox=None,
showfliers=None, boxprops=None, labels=None, flierprops=None, medianprops=None,
meanprops=None, capprops=None, whiskerprops=None, manage_ticks=True,
autorange=False, zorder=None, capwidths=None, *, data=None)
    Draw a box and whisker plot.

    The box extends from the first quartile (Q1) to the third
    quartile (Q3) of the data, with a line at the median.  The
    whiskers extend from the box by 1.5x the inter-quartile range
    (IQR).  Flier points are those past the end of the whiskers.
    See https://en.wikipedia.org/wiki/Box_plot for reference.

    .. code-block:: none

              Q1-1.5IQR   Q1   median  Q3   Q3+1.5IQR
                           |-----:-----|
           o      |--------|     :     |--------|    o  o
                           |-----:-----|
         flier             <----------->            fliers
                                IQR


    Parameters
    ----------
    x : Array or a sequence of vectors.
        The input data.  If a 2D array, a boxplot is drawn for each column
        in *x*.  If a sequence of 1D arrays, a boxplot is drawn for each
        array in *x*.

    notch : bool, default: False
        Whether to draw a notched boxplot (`True`), or a rectangular
        boxplot (`False`).  The notches represent the confidence interval
        (CI) around the median.  The documentation for *bootstrap*
        describes how the locations of the notches are computed by
        default, but their locations may also be overridden by setting the
        *conf_intervals* parameter.

        .. note::

            In cases where the values of the CI are less than the
            lower quartile or greater than the upper quartile, the
            notches will extend beyond the box, giving it a
            distinctive "flipped" appearance. This is expected
            behavior and consistent with other statistical
            visualization packages.

    sym : str, optional
        The default symbol for flier points.  An empty string ('') hides
        the fliers.  If `None`, then the fliers default to 'b+'.  More
        control is provided by the *flierprops* parameter.
```

```
vert : bool, default: True
    If `True`, draws vertical boxes.
    If `False`, draw horizontal boxes.

whis : float or (float, float), default: 1.5
    The position of the whiskers.

    If a float, the lower whisker is at the lowest datum above
    ``Q1 - whis*(Q3-Q1)``, and the upper whisker at the highest datum
    below ``Q3 + whis*(Q3-Q1)``, where Q1 and Q3 are the first and
    third quartiles.  The default value of ``whis = 1.5`` corresponds
    to Tukey's original definition of boxplots.

    If a pair of floats, they indicate the percentiles at which to
    draw the whiskers (e.g., (5, 95)).  In particular, setting this to
    (0, 100) results in whiskers covering the whole range of the data.

    In the edge case where ``Q1 == Q3``, *whis* is automatically set
    to (0, 100) (cover the whole range of the data) if *autorange* is
    True.

    Beyond the whiskers, data are considered outliers and are plotted
    as individual points.

bootstrap : int, optional
    Specifies whether to bootstrap the confidence intervals
    around the median for notched boxplots. If *bootstrap* is
    None, no bootstrapping is performed, and notches are
    calculated using a Gaussian-based asymptotic approximation
    (see McGill, R., Tukey, J.W., and Larsen, W.A., 1978, and
    Kendall and Stuart, 1967). Otherwise, bootstrap specifies
    the number of times to bootstrap the median to determine its
    95% confidence intervals. Values between 1000 and 10000 are
    recommended.

usermedians : 1D array-like, optional
    A 1D array-like of length ``len(x)``.  Each entry that is not
    `None` forces the value of the median for the corresponding
    dataset.  For entries that are `None`, the medians are computed
    by Matplotlib as normal.

conf_intervals : array-like, optional
    A 2D array-like of shape ``(len(x), 2)``.  Each entry that is not
    None forces the location of the corresponding notch (which is
    only drawn if *notch* is `True`).  For entries that are `None`,
    the notches are computed by the method specified by the other
    parameters (e.g., *bootstrap*).
```

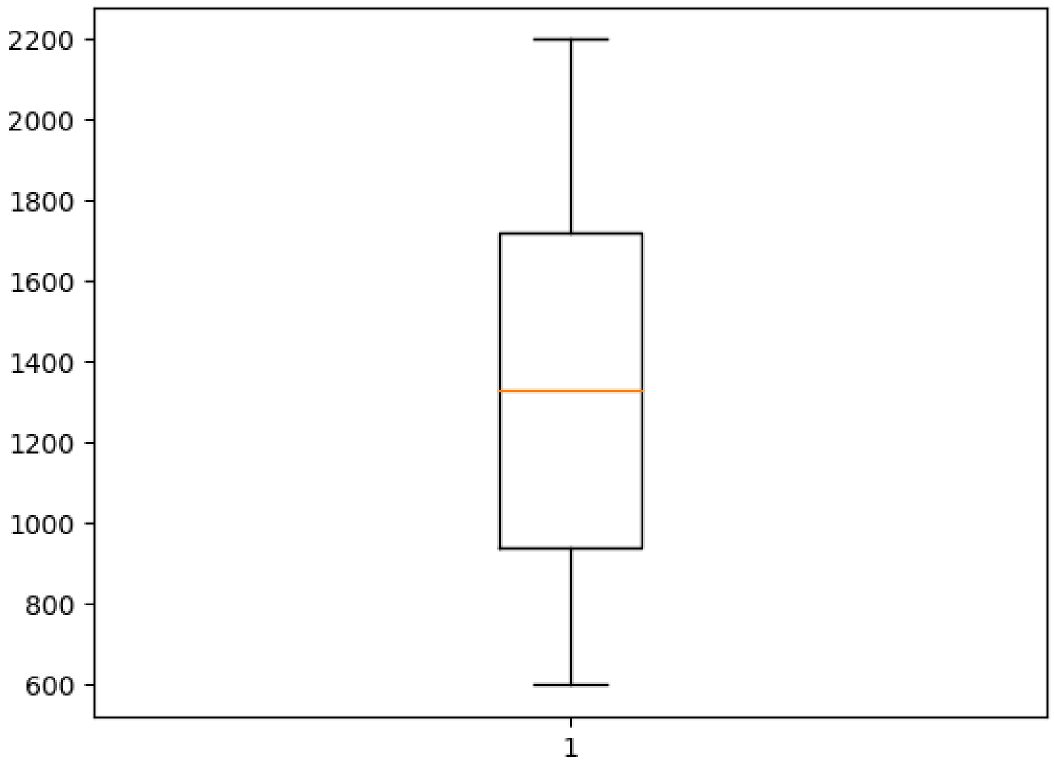
```
positions : array-like, optional
    The positions of the boxes. The ticks and limits are
    automatically set to match the positions. Defaults to
    ``range(1, N+1)`` where N is the number of boxes to be drawn.

widths : float or array-like
    The widths of the boxes.  The default is 0.5, or ``0.15*(distance
    between extreme positions)``, if that is smaller.

patch_artist : bool, default: False
    If `False` produces boxes with the Line2D artist. Otherwise,
    boxes are drawn with Patch artists.

labels : sequence, optional
    Labels for each dataset (one per dataset).

manage_ticks : bool, default: True
    If True, the tick locations and labels will be adjusted to match
    the boxplot positions.

autorange : bool, default: False
    When `True` and the data are distributed such that the 25th and
    75th percentiles are equal, *whis* is set to (0, 100) such
    that the whisker ends are at the minimum and maximum of the data.

meanline : bool, default: False
    If `True` (and *showmeans* is `True`), will try to render the
    mean as a line spanning the full width of the box according to
    *meanprops* (see below).  Not recommended if *shownotches* is also
    True.  Otherwise, means will be shown as points.

zorder : float, default: ``Line2D.zorder = 2``
    The zorder of the boxplot.

Returns
-------
dict
  A dictionary mapping each component of the boxplot to a list
  of the `.Line2D` instances created. That dictionary has the
  following keys (assuming vertical boxplots):

  - ``boxes``: the main body of the boxplot showing the
    quartiles and the median's confidence intervals if
    enabled.

  - ``medians``: horizontal lines at the median of each box.

  - ``whiskers``: the vertical lines extending to the most
```

```
    extreme, non-outlier data points.

  - ``caps``: the horizontal lines at the ends of the
    whiskers.

  - ``fliers``: points representing data that extend beyond
    the whiskers (fliers).

  - ``means``: points or lines representing the means.

Other Parameters
----------------
showcaps : bool, default: True
    Show the caps on the ends of whiskers.
showbox : bool, default: True
    Show the central box.
showfliers : bool, default: True
    Show the outliers beyond the caps.
showmeans : bool, default: False
    Show the arithmetic means.
capprops : dict, default: None
    The style of the caps.
capwidths : float or array, default: None
    The widths of the caps.
boxprops : dict, default: None
    The style of the box.
whiskerprops : dict, default: None
    The style of the whiskers.
flierprops : dict, default: None
    The style of the fliers.
medianprops : dict, default: None
    The style of the median.
meanprops : dict, default: None
    The style of the mean.
data : indexable object, optional
    If given, all parameters also accept a string ``s``, which is
    interpreted as ``data[s]`` (unless this raises an exception).

See Also
--------
violinplot : Draw an estimate of the probability density function.
```



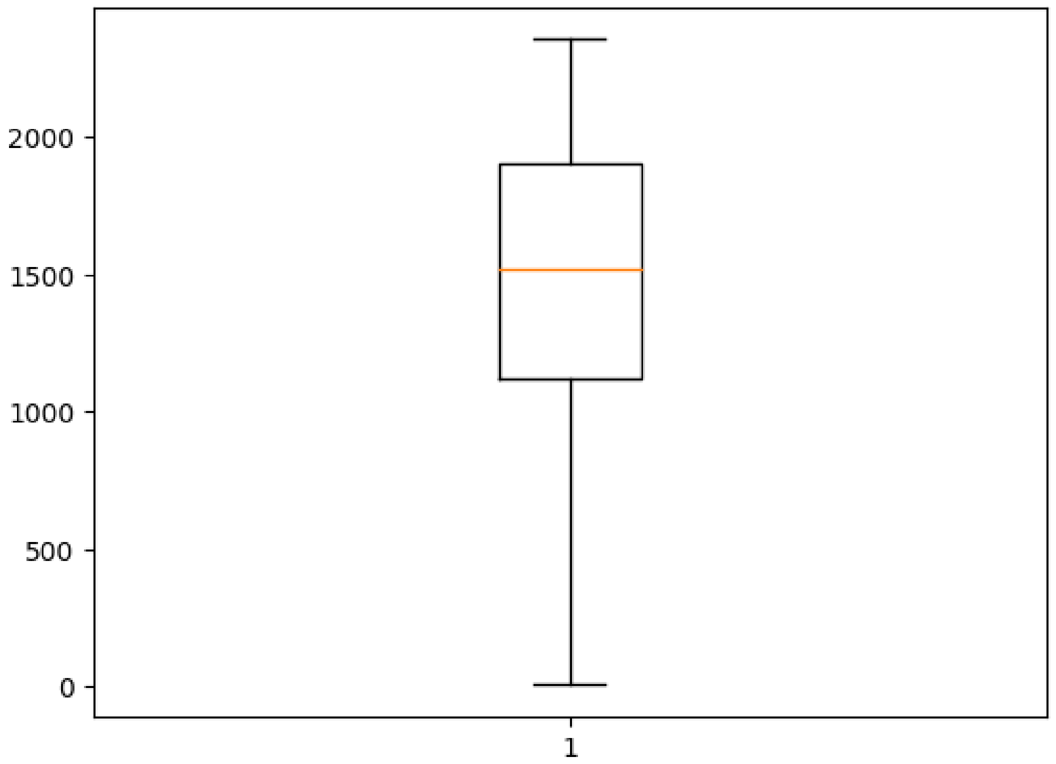
[38]:
```
plt.boxplot(project.Planned_Shipment_Time)
```

[38]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7fe053abfd00>,
  <matplotlib.lines.Line2D at 0x7fe053abffa0>],
 'caps': [<matplotlib.lines.Line2D at 0x7fe0538ec280>,
  <matplotlib.lines.Line2D at 0x7fe0538ec520>],
 'boxes': [<matplotlib.lines.Line2D at 0x7fe053abfa60>],
 'medians': [<matplotlib.lines.Line2D at 0x7fe0538ec7c0>],
 'fliers': [<matplotlib.lines.Line2D at 0x7fe0538eca60>],
 'means': []}
```



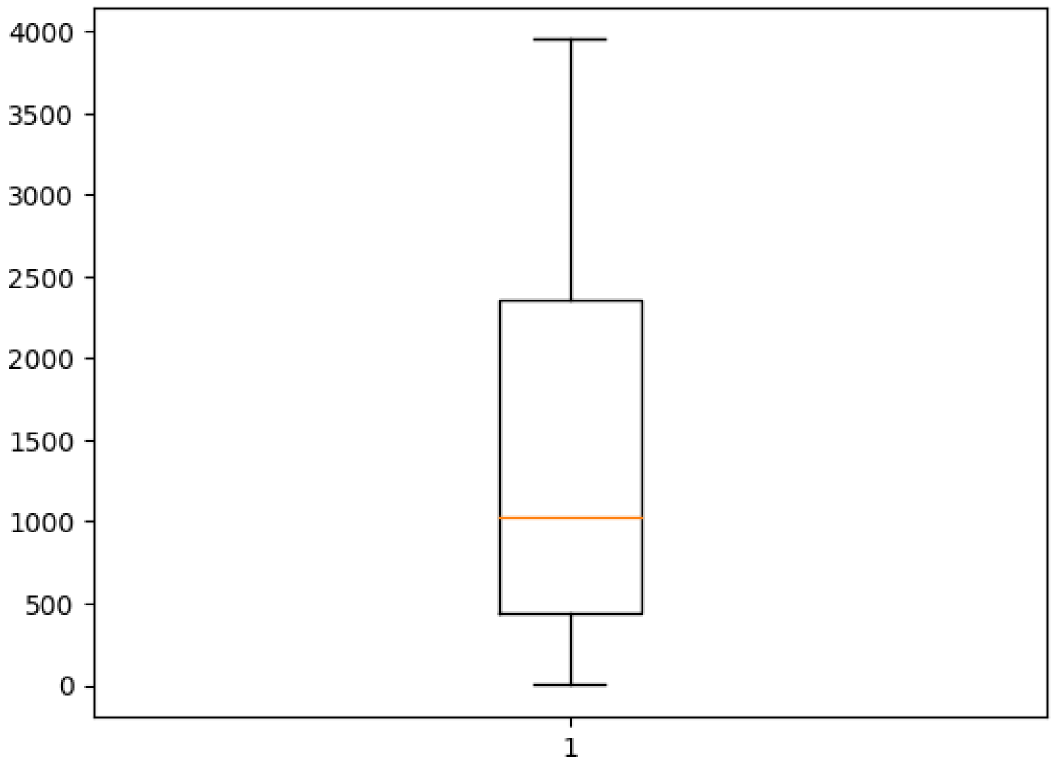
```
[39]: plt.boxplot(project.Planned_Delivery_Time)
```

```
[39]: {'whiskers': [<matplotlib.lines.Line2D at 0x7fe053946c50>,
        <matplotlib.lines.Line2D at 0x7fe053946ef0>],
       'caps': [<matplotlib.lines.Line2D at 0x7fe053947190>,
        <matplotlib.lines.Line2D at 0x7fe053947430>],
       'boxes': [<matplotlib.lines.Line2D at 0x7fe0539469b0>],
       'medians': [<matplotlib.lines.Line2D at 0x7fe0539476d0>],
       'fliers': [<matplotlib.lines.Line2D at 0x7fe053947970>],
       'means': []}
```



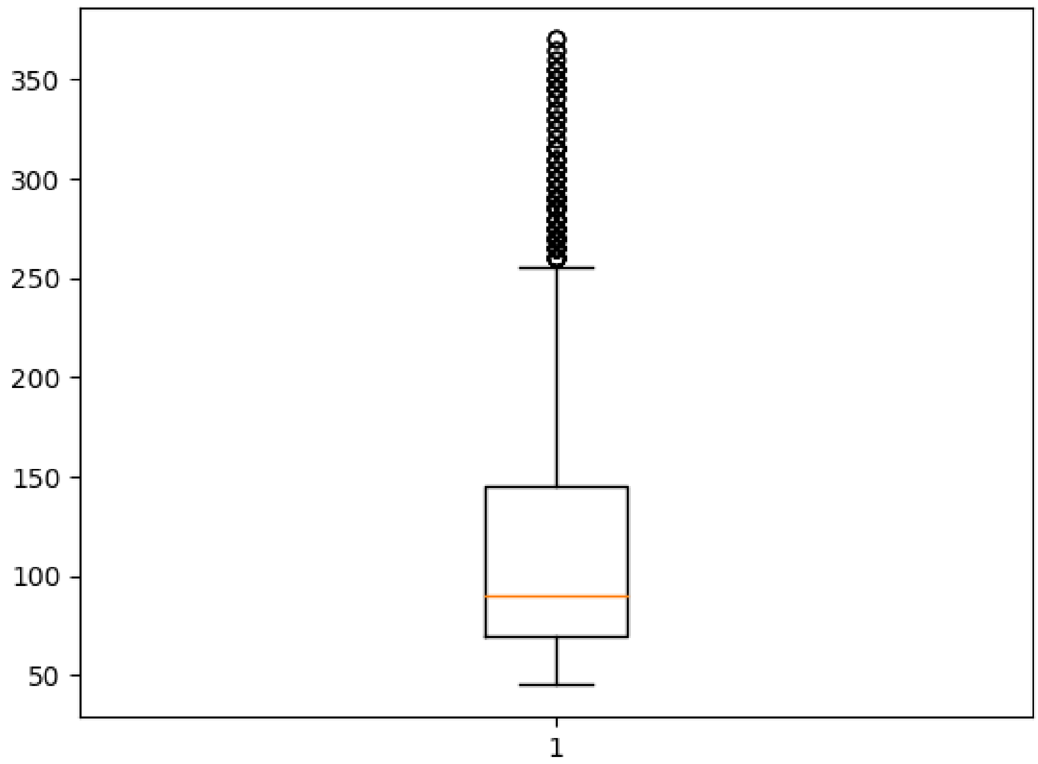
```
[40]: plt.boxplot(project.Carrier_Num)
```

```
[40]: {'whiskers': [<matplotlib.lines.Line2D at 0x7fe0539a2d10>,
        <matplotlib.lines.Line2D at 0x7fe0539d4160>],
       'caps': [<matplotlib.lines.Line2D at 0x7fe0539d4340>,
        <matplotlib.lines.Line2D at 0x7fe0539d45e0>],
       'boxes': [<matplotlib.lines.Line2D at 0x7fe0539a3f10>],
       'medians': [<matplotlib.lines.Line2D at 0x7fe0539d4880>],
       'fliers': [<matplotlib.lines.Line2D at 0x7fe0539d4b20>],
       'means': []}
```



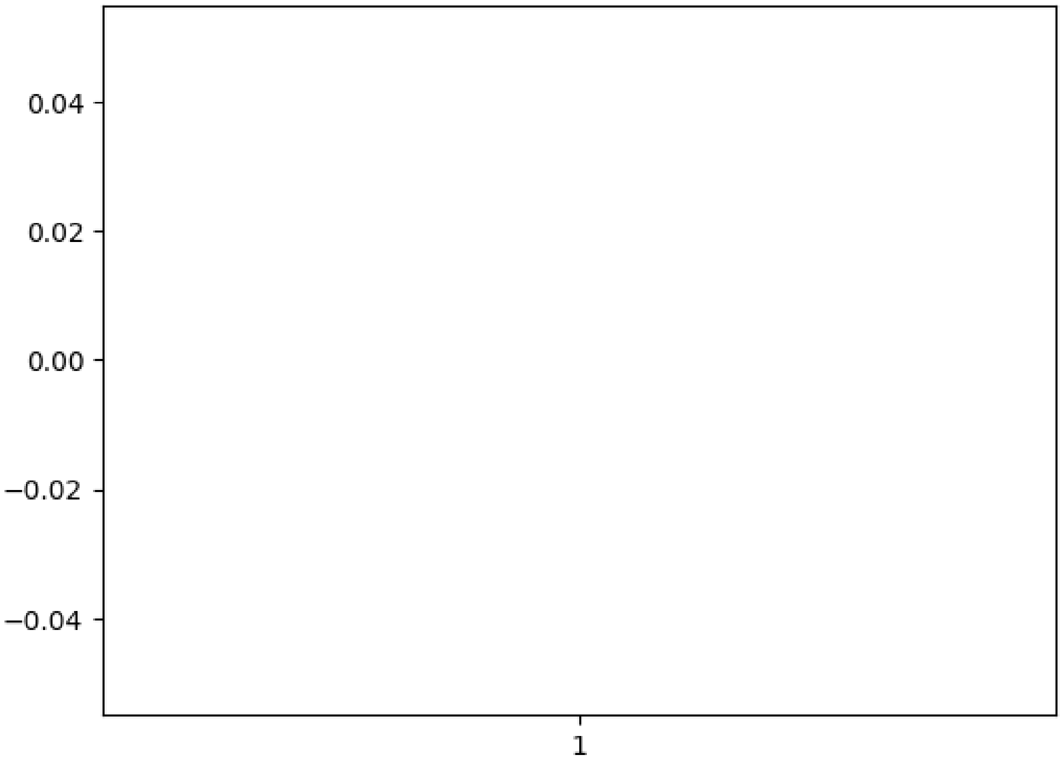
[41]:
```
plt.boxplot(project.Planned_TimeofTravel)
```

[41]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7fe053823130>,
  <matplotlib.lines.Line2D at 0x7fe0538233d0>],
 'caps': [<matplotlib.lines.Line2D at 0x7fe053823670>,
  <matplotlib.lines.Line2D at 0x7fe053823910>],
 'boxes': [<matplotlib.lines.Line2D at 0x7fe053822e90>],
 'medians': [<matplotlib.lines.Line2D at 0x7fe053823bb0>],
 'fliers': [<matplotlib.lines.Line2D at 0x7fe053823e50>],
 'means': []}
```



```
[42]: plt.boxplot(project.Shipment_Delay)
```

```
[42]: {'whiskers': [<matplotlib.lines.Line2D at 0x7fe0538a8be0>,
        <matplotlib.lines.Line2D at 0x7fe0538a8e80>],
       'caps': [<matplotlib.lines.Line2D at 0x7fe0538a9120>,
        <matplotlib.lines.Line2D at 0x7fe0538a93c0>],
       'boxes': [<matplotlib.lines.Line2D at 0x7fe0539d59f0>],
       'medians': [<matplotlib.lines.Line2D at 0x7fe0538a9660>],
       'fliers': [<matplotlib.lines.Line2D at 0x7fe0538a9900>],
       'means': []}
```



[43]:

```
help(plt.boxplot)
```

```
Help on function boxplot in module matplotlib.pyplot:

boxplot(x, notch=None, sym=None, vert=None, whis=None, positions=None,
widths=None, patch_artist=None, bootstrap=None, usermedians=None,
conf_intervals=None, meanline=None, showmeans=None, showcaps=None, showbox=None,
showfliers=None, boxprops=None, labels=None, flierprops=None, medianprops=None,
meanprops=None, capprops=None, whiskerprops=None, manage_ticks=True,
autorange=False, zorder=None, capwidths=None, *, data=None)
    Draw a box and whisker plot.

    The box extends from the first quartile (Q1) to the third
    quartile (Q3) of the data, with a line at the median.  The
    whiskers extend from the box by 1.5x the inter-quartile range
    (IQR).  Flier points are those past the end of the whiskers.
    See https://en.wikipedia.org/wiki/Box_plot for reference.

    .. code-block:: none

              Q1-1.5IQR   Q1   median  Q3   Q3+1.5IQR
                           |-----:-----|
           o      |--------|     :     |--------|    o  o
                           |-----:-----|
         flier             <----------->            fliers
                                IQR


    Parameters
    ----------
    x : Array or a sequence of vectors.
        The input data.  If a 2D array, a boxplot is drawn for each column
        in *x*.  If a sequence of 1D arrays, a boxplot is drawn for each
        array in *x*.

    notch : bool, default: False
        Whether to draw a notched boxplot (`True`), or a rectangular
        boxplot (`False`).  The notches represent the confidence interval
        (CI) around the median.  The documentation for *bootstrap*
        describes how the locations of the notches are computed by
        default, but their locations may also be overridden by setting the
        *conf_intervals* parameter.

        .. note::

            In cases where the values of the CI are less than the
            lower quartile or greater than the upper quartile, the
            notches will extend beyond the box, giving it a
            distinctive "flipped" appearance. This is expected
            behavior and consistent with other statistical
            visualization packages.

    sym : str, optional
        The default symbol for flier points.  An empty string ('') hides
        the fliers.  If `None`, then the fliers default to 'b+'.  More
        control is provided by the *flierprops* parameter.

    vert : bool, default: True
        If `True`, draws vertical boxes.
        If `False`, draw horizontal boxes.

    whis : float or (float, float), default: 1.5
        The position of the whiskers.

        If a float, the lower whisker is at the lowest datum above
        ``Q1 - whis*(Q3-Q1)``, and the upper whisker at the highest datum
        below ``Q3 + whis*(Q3-Q1)``, where Q1 and Q3 are the first and
        third quartiles.  The default value of ``whis = 1.5`` corresponds
        to Tukey's original definition of boxplots.
        If a pair of floats, they indicate the percentiles at which to
        draw the whiskers (e.g., (5, 95)).  In particular, setting this to
        (0, 100) results in whiskers covering the whole range of the data.

        In the edge case where ``Q1 == Q3``, *whis* is automatically set
        to (0, 100) (cover the whole range of the data) if *autorange* is
        True.

        Beyond the whiskers, data are considered outliers and are plotted
        as individual points.

    bootstrap : int, optional
        Specifies whether to bootstrap the confidence intervals
        around the median for notched boxplots. If *bootstrap* is
        None, no bootstrapping is performed, and notches are
        calculated using a Gaussian-based asymptotic approximation
        (see McGill, R., Tukey, J.W., and Larsen, W.A., 1978, and
        Kendall and Stuart, 1967). Otherwise, bootstrap specifies
        the number of times to bootstrap the median to determine its
        95% confidence intervals. Values between 1000 and 10000 are
        recommended.

    usermedians : 1D array-like, optional
        A 1D array-like of length ``len(x)``.  Each entry that is not
        `None` forces the value of the median for the corresponding
        dataset.  For entries that are `None`, the medians are computed
        by Matplotlib as normal.

    conf_intervals : array-like, optional
        A 2D array-like of shape ``(len(x), 2)``.  Each entry that is not
        None forces the location of the corresponding notch (which is
        only drawn if *notch* is `True`).  For entries that are `None`,
        the notches are computed by the method specified by the other
        parameters (e.g., *bootstrap*).

    positions : array-like, optional
        The positions of the boxes. The ticks and limits are
        automatically set to match the positions. Defaults to
        ``range(1, N+1)`` where N is the number of boxes to be drawn.

    widths : float or array-like
        The widths of the boxes.  The default is 0.5, or ``0.15*(distance
        between extreme positions)``, if that is smaller.

    patch_artist : bool, default: False
        If `False` produces boxes with the Line2D artist. Otherwise,
        boxes are drawn with Patch artists.
```

```
labels : sequence, optional
    Labels for each dataset (one per dataset).

manage_ticks : bool, default: True
    If True, the tick locations and labels will be adjusted to match
    the boxplot positions.

autorange : bool, default: False
    When `True` and the data are distributed such that the 25th and
    75th percentiles are equal, *whis* is set to (0, 100) such
    that the whisker ends are at the minimum and maximum of the data.

meanline : bool, default: False
    If `True` (and *showmeans* is `True`), will try to render the
    mean as a line spanning the full width of the box according to
    *meanprops* (see below).  Not recommended if *shownotches* is also
    True.  Otherwise, means will be shown as points.

zorder : float, default: ``Line2D.zorder = 2``
    The zorder of the boxplot.

Returns
-------
dict
  A dictionary mapping each component of the boxplot to a list
  of the `.Line2D` instances created. That dictionary has the
  following keys (assuming vertical boxplots):

  - ``boxes``: the main body of the boxplot showing the
    quartiles and the median's confidence intervals if
    enabled.

  - ``medians``: horizontal lines at the median of each box.

  - ``whiskers``: the vertical lines extending to the most
    extreme, non-outlier data points.

  - ``caps``: the horizontal lines at the ends of the
    whiskers.

  - ``fliers``: points representing data that extend beyond
    the whiskers (fliers).

  - ``means``: points or lines representing the means.

Other Parameters
----------------
showcaps : bool, default: True
```

```
        Show the caps on the ends of whiskers.
    showbox : bool, default: True
        Show the central box.
    showfliers : bool, default: True
        Show the outliers beyond the caps.
    showmeans : bool, default: False
        Show the arithmetic means.
    capprops : dict, default: None
        The style of the caps.
    capwidths : float or array, default: None
        The widths of the caps.
    boxprops : dict, default: None
        The style of the box.
    whiskerprops : dict, default: None
        The style of the whiskers.
    flierprops : dict, default: None
        The style of the fliers.
    medianprops : dict, default: None
        The style of the median.
    meanprops : dict, default: None
        The style of the mean.
    data : indexable object, optional
        If given, all parameters also accept a string ``s``, which is
        interpreted as ``data[s]`` (unless this raises an exception).

    See Also
    --------
    violinplot : Draw an estimate of the probability density function.
```

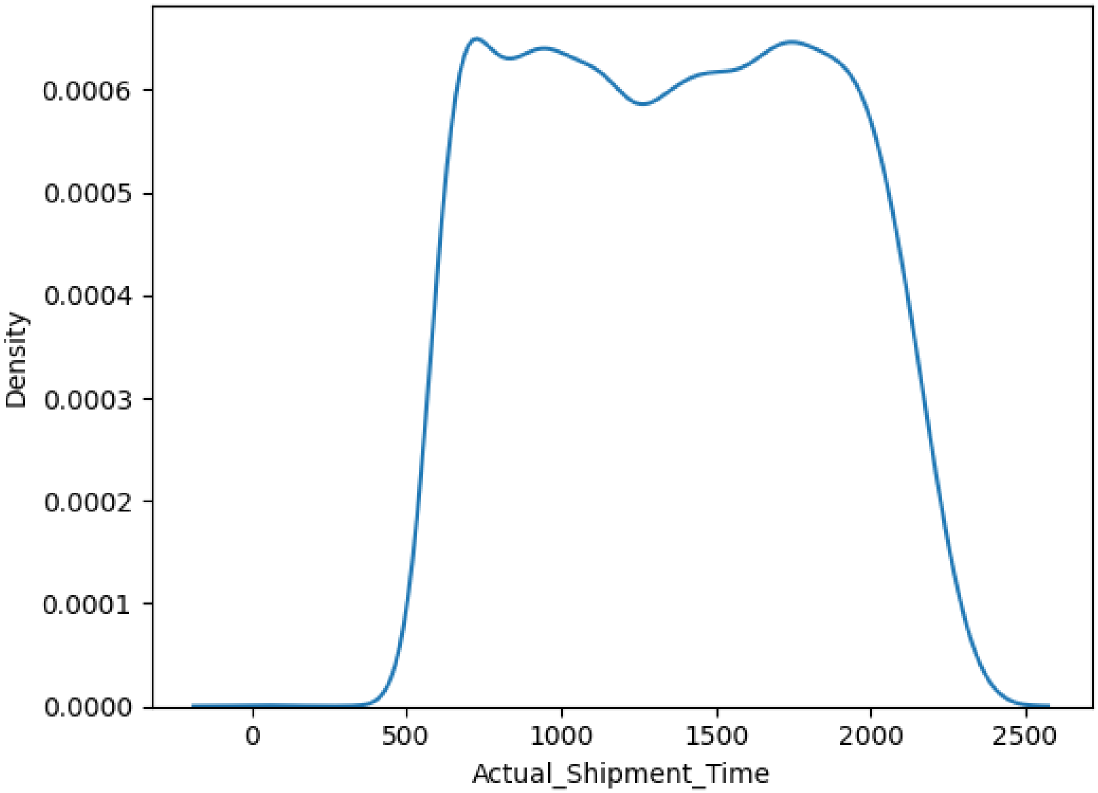
8 Density Plot

[44]:

```
sns.kdeplot(project.Actual_Shipment_Time)
```

[44]:

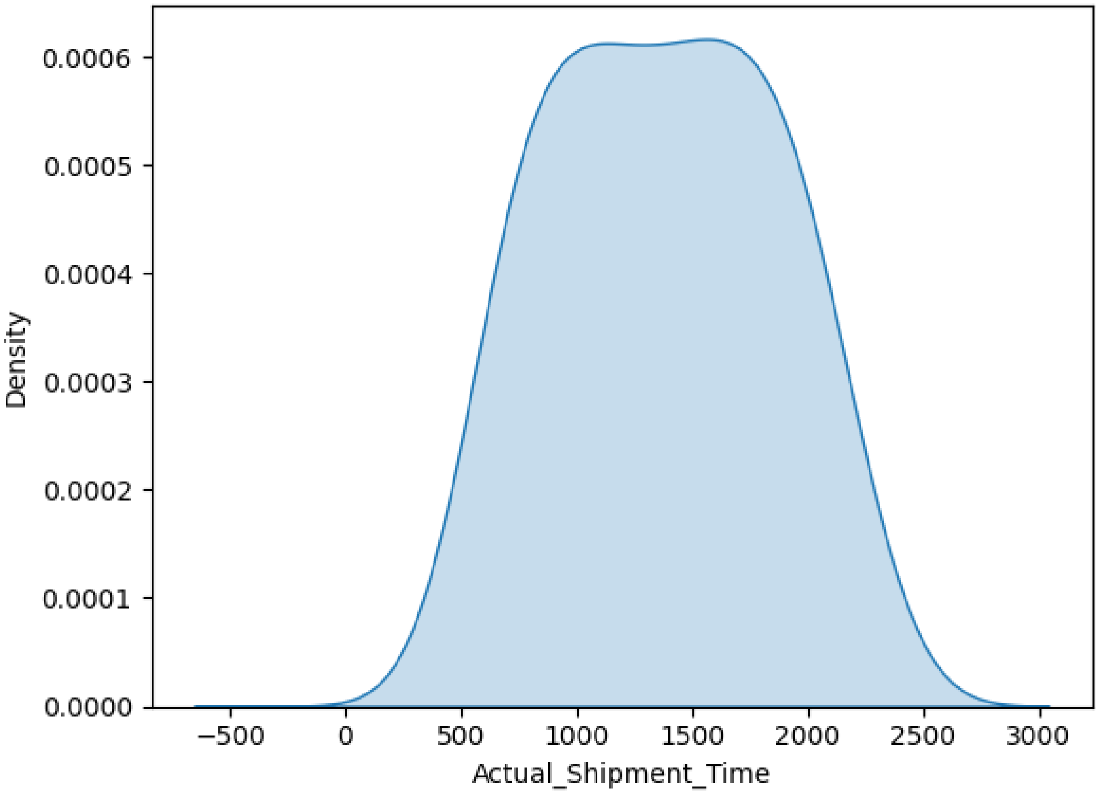
```
<Axes: xlabel='Actual_Shipment_Time', ylabel='Density'>
```



```
[45]: sns.kdeplot(project.Actual_Shipment_Time, bw = 0.5 , fill = True)
```

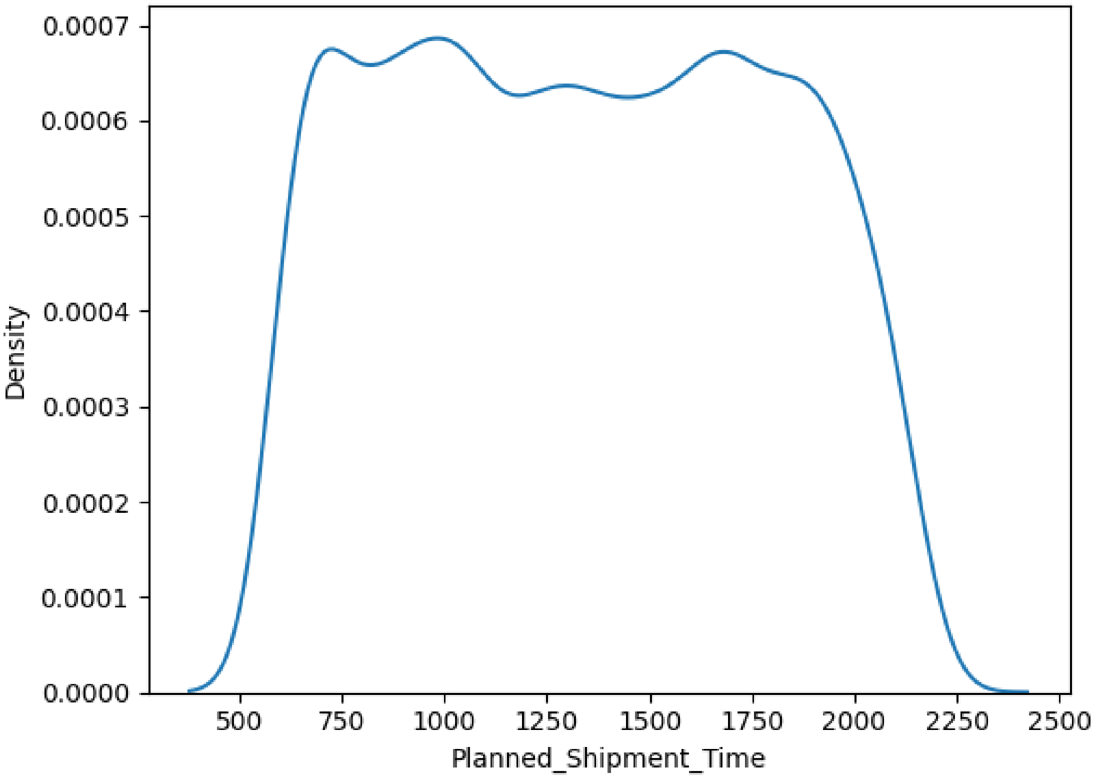
```
<ipython-input-45-5a6e0bd28785>:1: UserWarning:

The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`.
Setting `bw_method=0.5`, but please see the docs for the new parameters
and update your code. This will become an error in seaborn v0.13.0.

  sns.kdeplot(project.Actual_Shipment_Time, bw = 0.5 , fill = True)
```

```
[45]: <Axes: xlabel='Actual_Shipment_Time', ylabel='Density'>
```



[46]:
```
sns.kdeplot(project.Planned_Shipment_Time)
```

[46]:
```
<Axes: xlabel='Planned_Shipment_Time', ylabel='Density'>
```



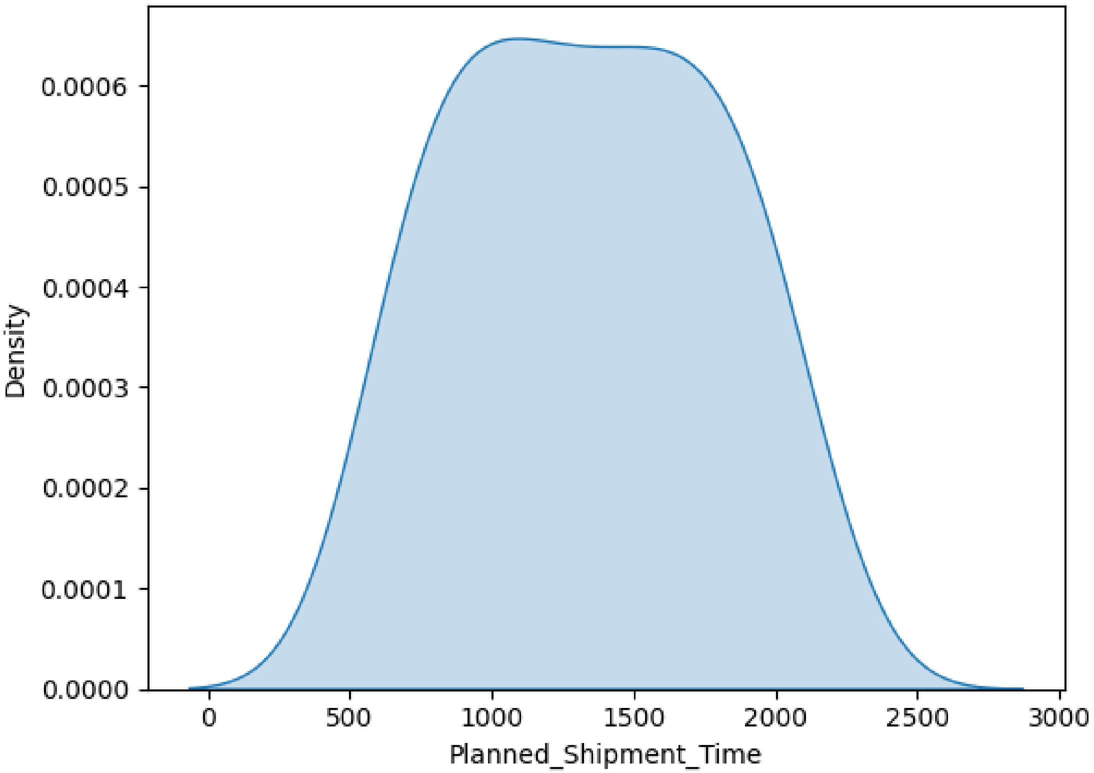
[47]:

```
sns.kdeplot(project.Planned_Shipment_Time, bw = 0.5 , fill = True)
```

```
<ipython-input-47-14589e1aab3d>:1: UserWarning:

The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`.
Setting `bw_method=0.5`, but please see the docs for the new parameters
and update your code. This will become an error in seaborn v0.13.0.

  sns.kdeplot(project.Planned_Shipment_Time, bw = 0.5 , fill = True)
```

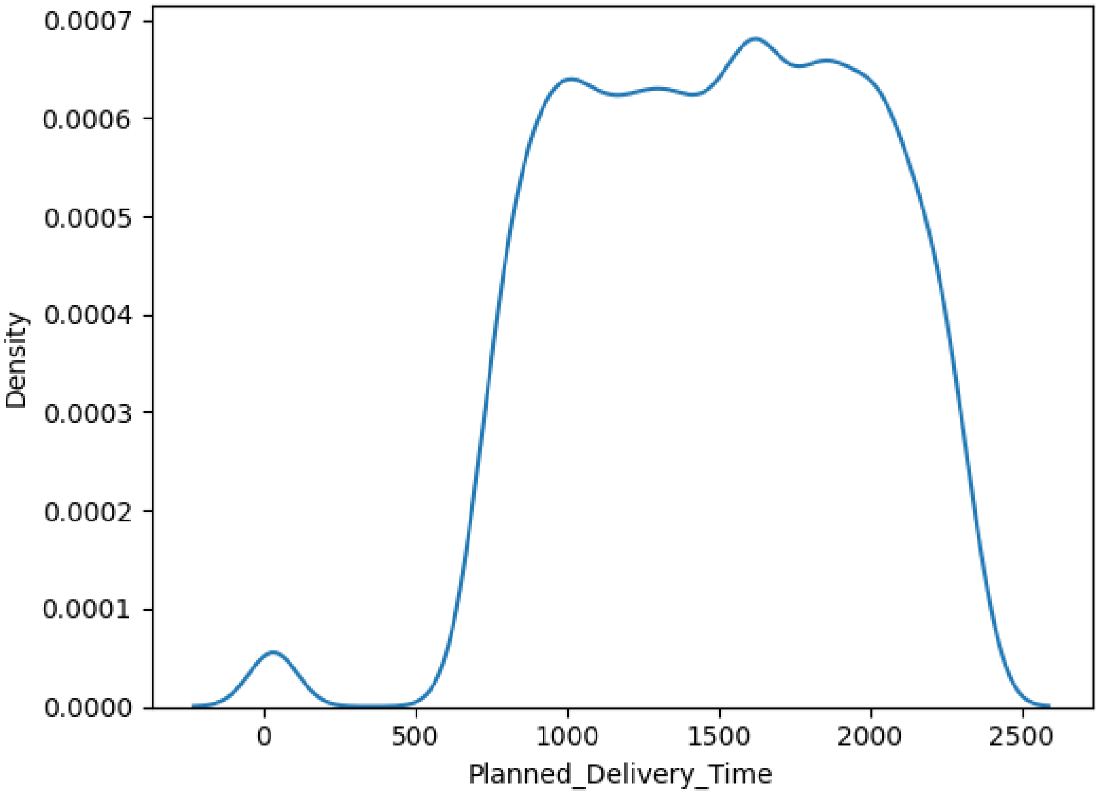
[47]:

```
<Axes: xlabel='Planned_Shipment_Time', ylabel='Density'>
```



[48]:
```
sns.kdeplot(project.Planned_Delivery_Time)
```

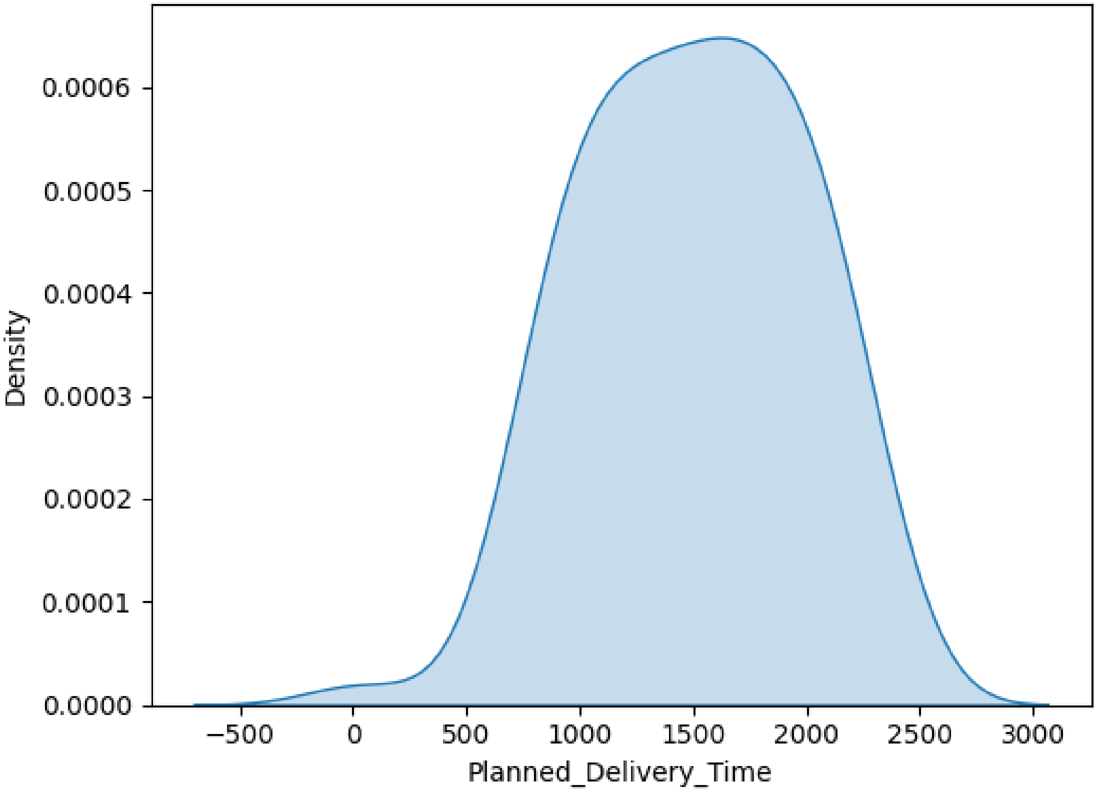
[48]:
```
<Axes: xlabel='Planned_Delivery_Time', ylabel='Density'>
```



[49]:
```
sns.kdeplot(project.Planned_Delivery_Time, bw = 0.5 , fill = True)
```

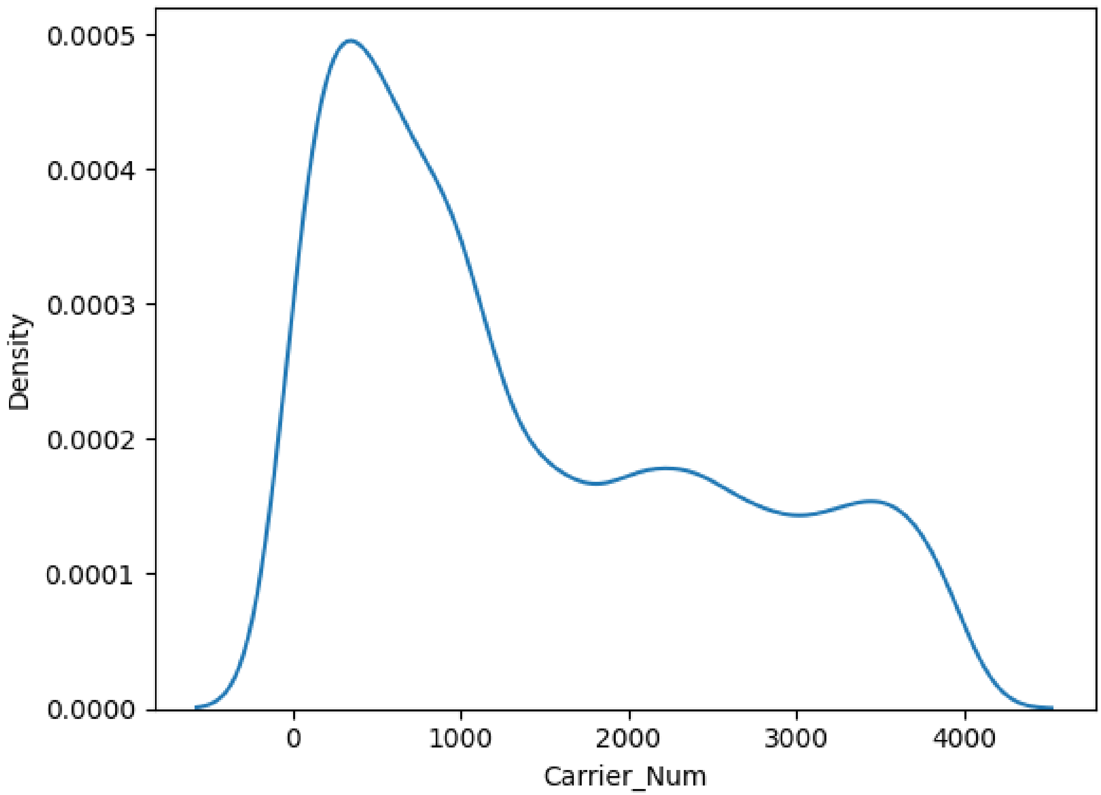
```
<ipython-input-49-87f25f9ee559>:1: UserWarning:

The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`.
Setting `bw_method=0.5`, but please see the docs for the new parameters
and update your code. This will become an error in seaborn v0.13.0.

  sns.kdeplot(project.Planned_Delivery_Time, bw = 0.5 , fill = True)
```

[49]:
```
<Axes: xlabel='Planned_Delivery_Time', ylabel='Density'>
```



[50]:
```
sns.kdeplot(project.Carrier_Num)
```

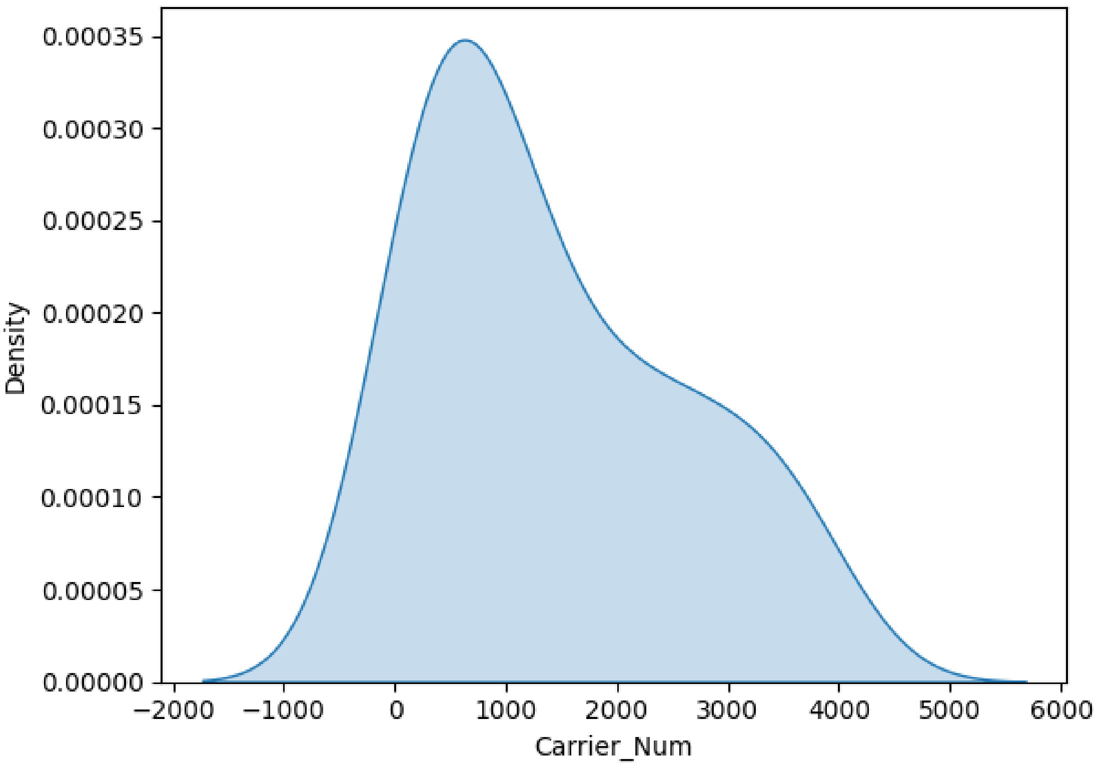
[50]:
```
<Axes: xlabel='Carrier_Num', ylabel='Density'>
```



```
[51]: sns.kdeplot(project.Carrier_Num, bw = 0.5 , fill = True)
```

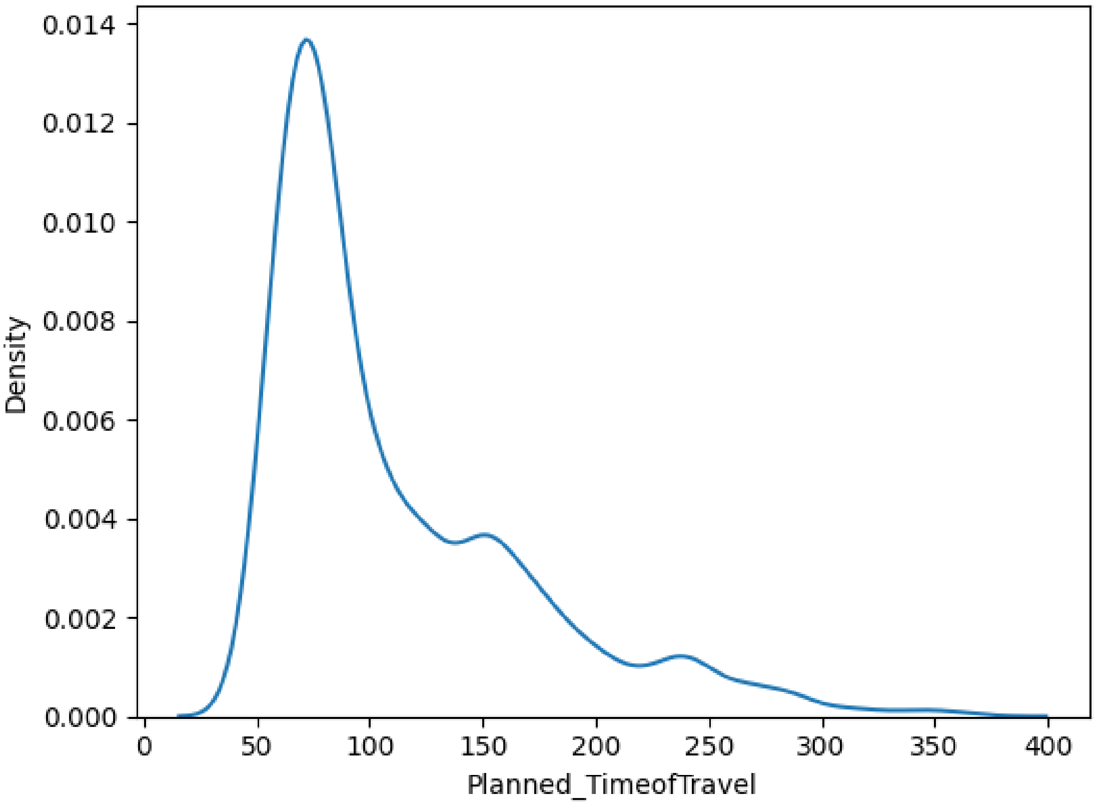
```
<ipython-input-51-e98e31b74cd8>:1: UserWarning:

The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`.
Setting `bw_method=0.5`, but please see the docs for the new parameters
and update your code. This will become an error in seaborn v0.13.0.

  sns.kdeplot(project.Carrier_Num, bw = 0.5 , fill = True)
```

```
[51]: <Axes: xlabel='Carrier_Num', ylabel='Density'>
```



```
[52]: sns.kdeplot(project.Planned_TimeofTravel)
```

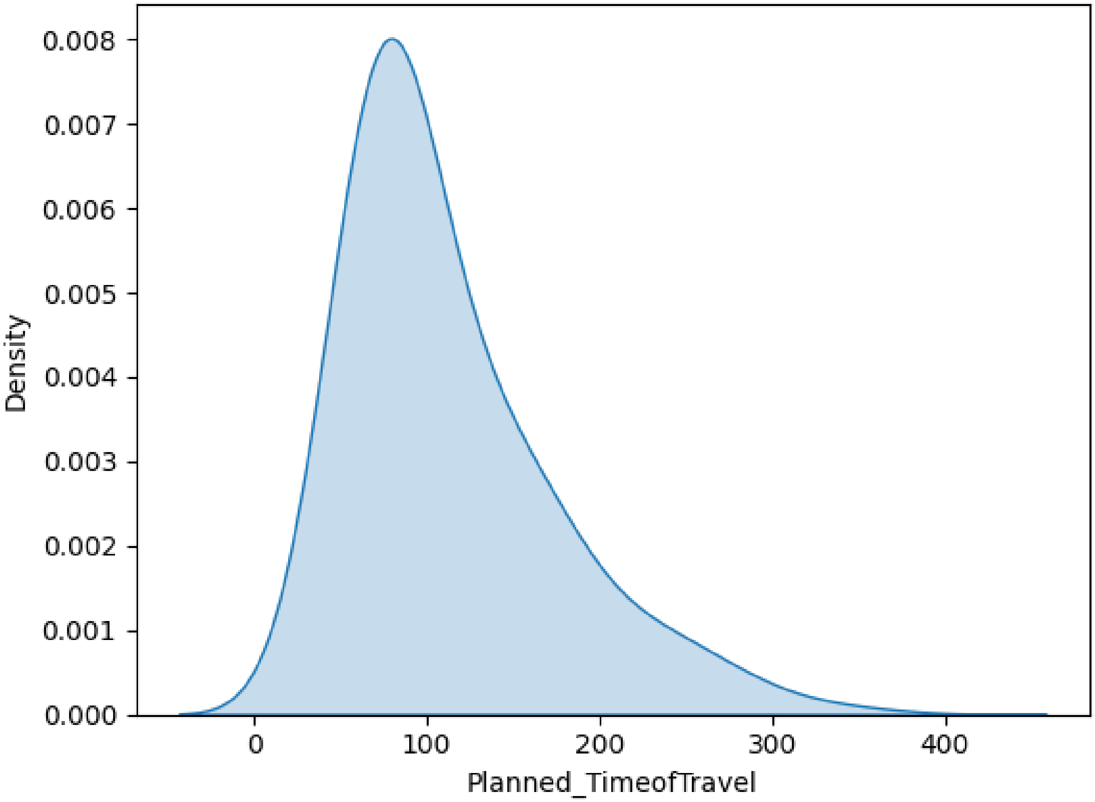
```
[52]: <Axes: xlabel='Planned_TimeofTravel', ylabel='Density'>
```



```
[53]: sns.kdeplot(project.Planned_TimeofTravel, bw = 0.5 , fill = True)
```

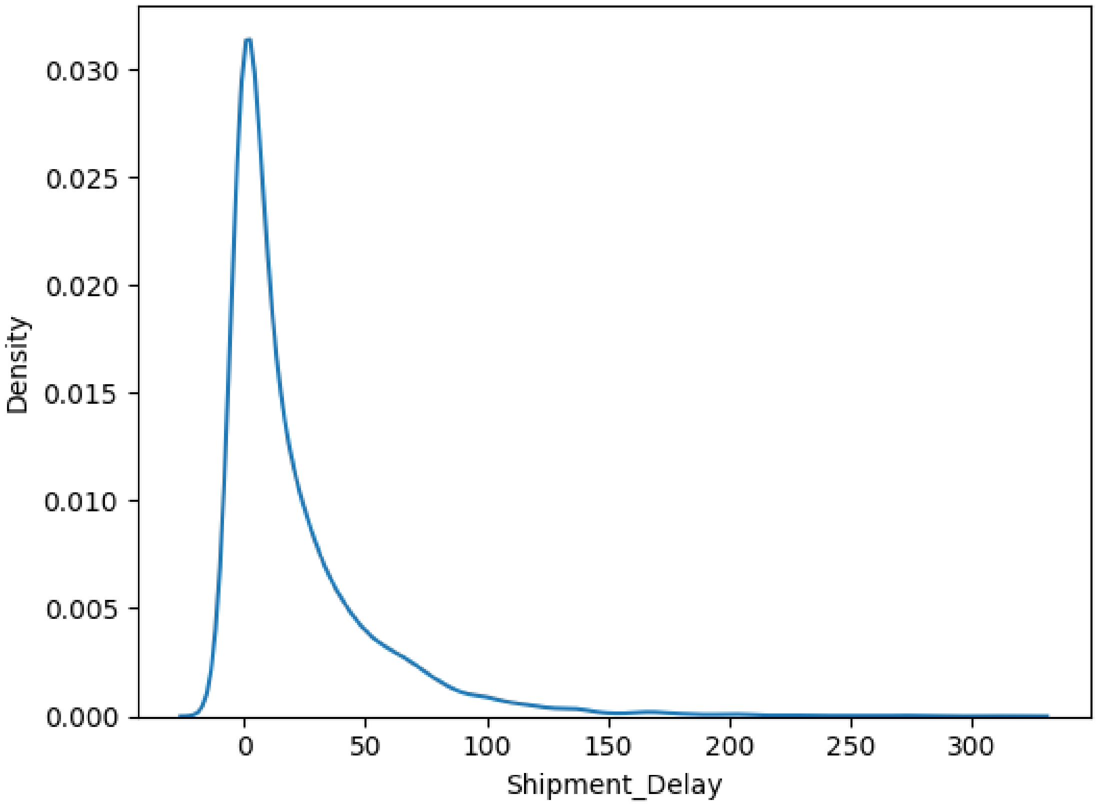
```
<ipython-input-53-80c2458b5a2e>:1: UserWarning:

The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`.
Setting `bw_method=0.5`, but please see the docs for the new parameters
and update your code. This will become an error in seaborn v0.13.0.

  sns.kdeplot(project.Planned_TimeofTravel, bw = 0.5 , fill = True)
```

```
[53]: <Axes: xlabel='Planned_TimeofTravel', ylabel='Density'>
```



```
[54]: sns.kdeplot(project.Shipment_Delay)
```

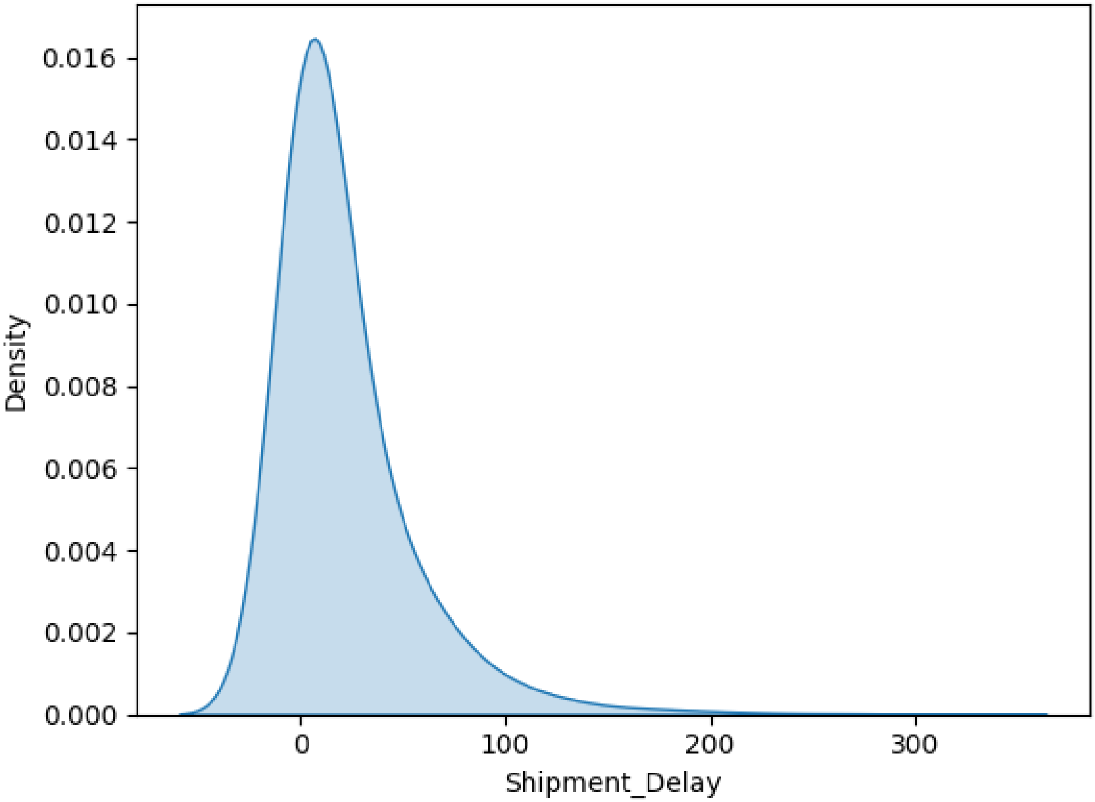
```
[54]: <Axes: xlabel='Shipment_Delay', ylabel='Density'>
```



[55]:
```
sns.kdeplot(project.Shipment_Delay, bw = 0.5 , fill = True)
```

```
<ipython-input-55-e8dac5fe4c2c>:1: UserWarning:

The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`.
Setting `bw_method=0.5`, but please see the docs for the new parameters
and update your code. This will become an error in seaborn v0.13.0.

  sns.kdeplot(project.Shipment_Delay, bw = 0.5 , fill = True)
```

[55]:
```
<Axes: xlabel='Shipment_Delay', ylabel='Density'>
```



9 Descriptive Statistics

10 describe function will return descriptive statistics including the

11 central tendency, dispersion and shape of a dataset's distribution.

```
[56]: project.describe()
```

```
[56]:         Year   Month   DayofMonth    DayOfWeek  Actual_Shipment_Time  \
count  7999.0  7999.0  7999.000000  7999.000000           7860.000000
mean   2008.0     1.0     3.978372     4.978372           1370.203435
std       0.0     0.0     0.754851     0.754851            468.043601
min    2008.0     1.0     3.000000     4.000000             47.000000
25%    2008.0     1.0     3.000000     4.000000            947.000000
50%    2008.0     1.0     4.000000     5.000000           1356.000000
75%    2008.0     1.0     5.000000     6.000000           1754.000000
max    2008.0     1.0     5.000000     6.000000           2341.000000
```

```
       Planned_Shipment_Time  Planned_Delivery_Time  Carrier_Num  \
count            7999.000000            7999.000000  7999.000000   
mean             1335.317540            1498.255407  1422.283285   
std               446.151375             473.788941  1155.282332   
min               600.000000               5.000000     1.000000   
25%               940.000000            1120.000000   445.500000   
50%              1330.000000            1520.000000  1023.000000   
75%              1720.000000            1905.000000  2358.500000   
max              2200.000000            2355.000000  3949.000000   

       Planned_TimeofTravel  Shipment_Delay     Distance  Delivery_Status  
count           7999.000000     7860.000000  7999.000000      7860.000000  
mean             112.899112       21.389186   637.847231         0.397074  
std               58.766090       32.563453   451.952916         0.489323  
min               45.000000      -10.000000   133.000000         0.000000  
25%               70.000000        1.000000   325.000000         0.000000  
50%               90.000000        9.000000   447.000000         0.000000  
75%              145.000000       30.000000   861.000000         1.000000  
max              370.000000      315.000000  2363.000000         1.000000  
```

12 Bivariate visualization

13 Scatter plot

[58]:

```
import pandas as pd
```

[57]:

```
import matplotlib.pyplot as plt
```

[59]:

```
project = pd.read_csv(r"/content/Datasets.csv")
```

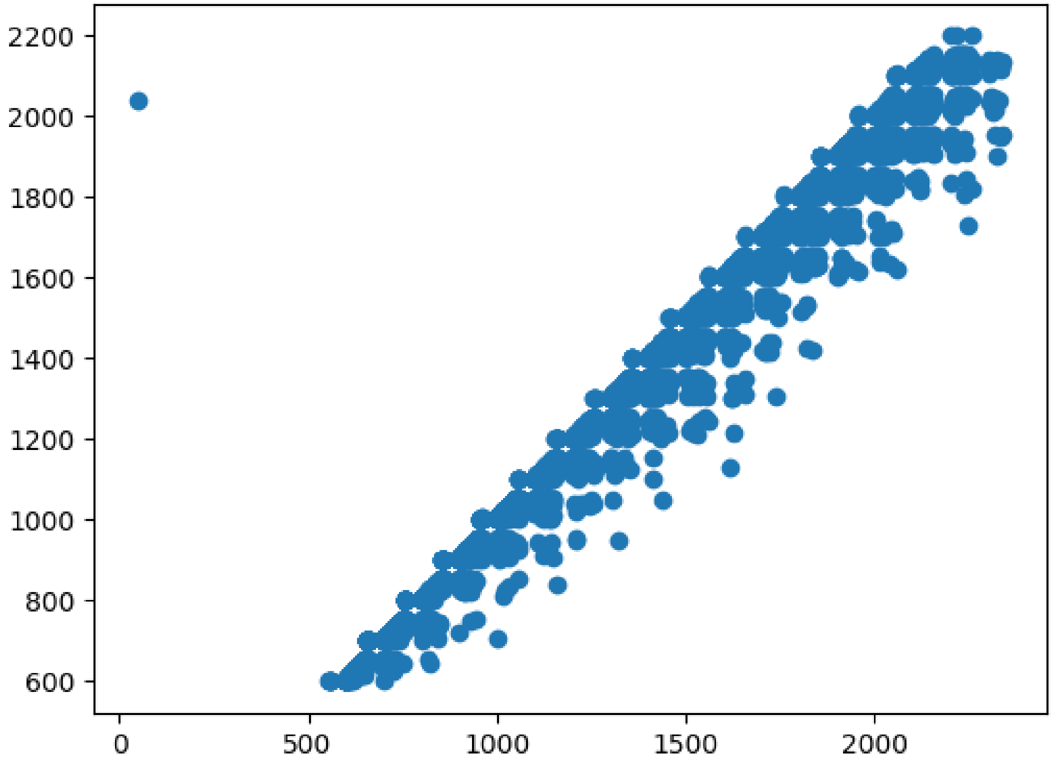
[60]:

```
project.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7999 entries, 0 to 7998
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Year                   7999 non-null   int64  
 1   Month                  7999 non-null   int64  
 2   DayofMonth             7999 non-null   int64  
 3   DayOfWeek              7999 non-null   int64  
 4   Actual_Shipment_Time   7860 non-null   float64
 5   Planned_Shipment_Time  7999 non-null   int64  
 6   Planned_Delivery_Time  7999 non-null   int64  
 7   Carrier_Name           7999 non-null   object 
 8   Carrier_Num            7999 non-null   int64  
 9   Planned_TimeofTravel   7999 non-null   int64
 10  Shipment_Delay         7860 non-null   float64
 11  Source                 7999 non-null   object
 12  Destination            7999 non-null   object
 13  Distance               7999 non-null   int64
 14  Delivery_Status        7860 non-null   float64
dtypes: float64(3), int64(9), object(3)
memory usage: 937.5+ KB
```

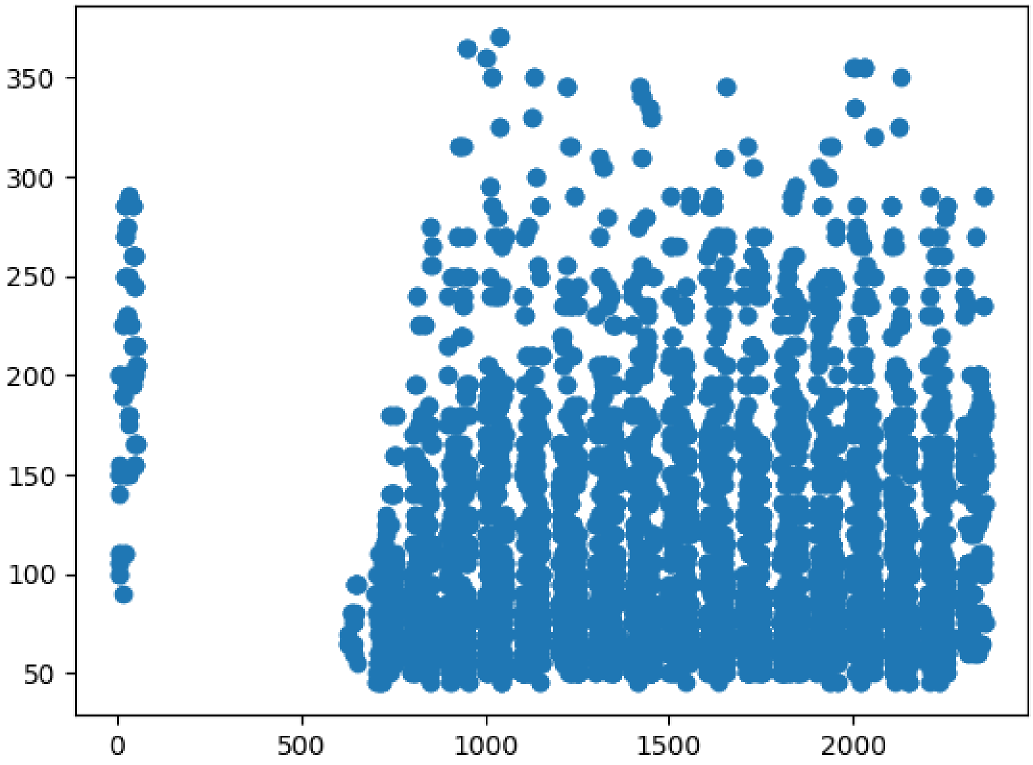
```
[61]: plt.scatter(x = project['Actual_Shipment_Time'], y =␣
       ↪project['Planned_Shipment_Time'])
```

```
[61]: <matplotlib.collections.PathCollection at 0x7fe0530b2710>
```



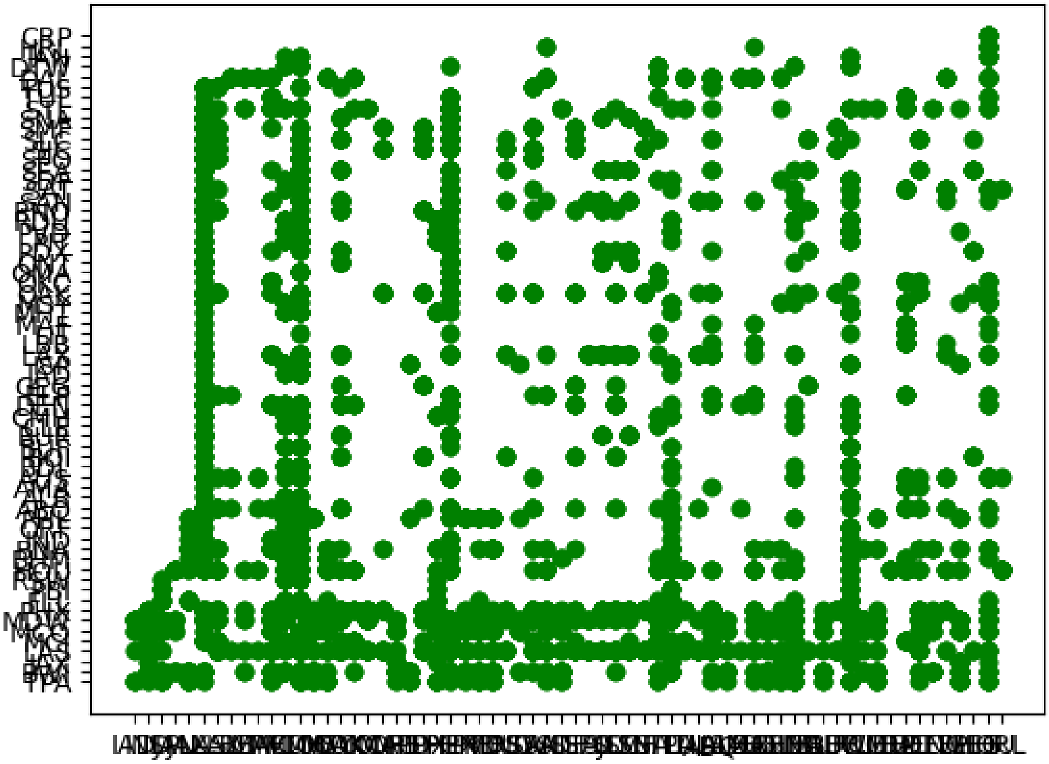
```
[62]: plt.scatter(x = project['Planned_Delivery_Time'], y =␣
       ↪project['Planned_TimeofTravel'])
```

```
[62]: <matplotlib.collections.PathCollection at 0x7fe052f33f70>
```



```
[63]: plt.scatter(x = project['Source'], y = project['Destination'], color = 'green')
```

```
[63]: <matplotlib.collections.PathCollection at 0x7fe052faefe0>
```


CRP
TPA