Working With Missing Data

References:

General techniques

https://pandas.pydata.org/pandas-docs/stable/missing_data.html

Missing Values in a Timeseries

https://www.kaggle.com/juejuewang/handle-missing-values-in-time-series-for-beginners

In [86]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

In [87]:
```
df = pd.read_csv('VehicleTraffic.csv', parse_dates=[0], index_col=0)
```

In [88]:
```
# Measurements taken at different times
df
```

Out[88]:

	Vehicles	Average Speed (mph)	Accidents
TimeStamp			
2018-12-04 13:00:00	95.0	38.0	0.0
2018-12-04 14:00:00	90.0	32.0	1.0
2018-12-04 15:00:00	98.0	30.0	1.0
2018-12-04 16:00:00	98.0	26.0	3.0
2018-12-04 17:00:00	NaN	NaN	NaN
2018-12-04 18:00:00	NaN	NaN	NaN
2018-12-04 19:00:00	84.0	35.0	2.0
2018-12-04 20:00:00	82.0	40.0	0.0
2018-12-04 21:00:00	77.0	45.0	0.0
2018-12-04 22:00:00	93.0	45.0	1.0

In [89]:
```
# Remove NaN values
df.dropna()
```

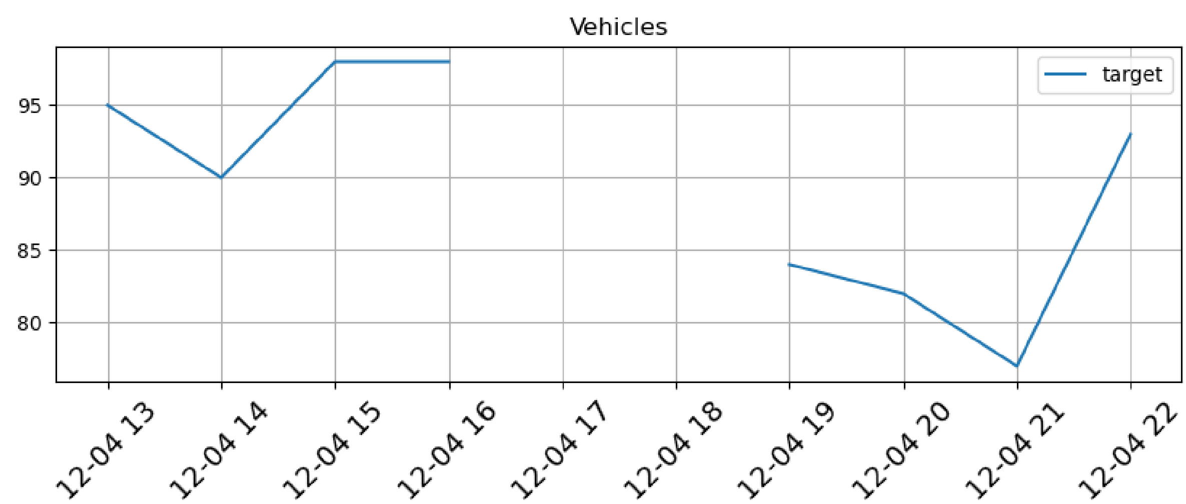
Out[89]:

TimeStamp	Vehicles	Average Speed (mph)	Accidents
2018-12-04 13:00:00	95.0	38.0	0.0
2018-12-04 14:00:00	90.0	32.0	1.0
2018-12-04 15:00:00	98.0	30.0	1.0
2018-12-04 16:00:00	98.0	26.0	3.0
2018-12-04 19:00:00	84.0	35.0	2.0
2018-12-04 20:00:00	82.0	40.0	0.0
2018-12-04 21:00:00	77.0	45.0	0.0
2018-12-04 22:00:00	93.0	45.0	1.0

In [90]:
```
# Mean values of numeric columns
df.mean()
```

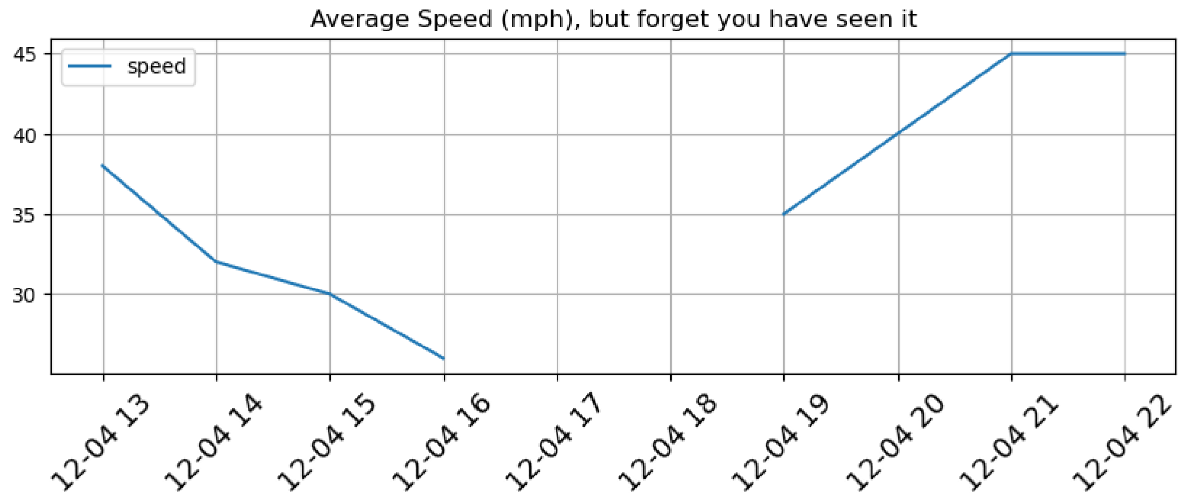
Out[90]:
```
Vehicles               89.625
Average Speed (mph)    36.375
Accidents               1.000
dtype: float64
```

In [91]:
```
# Let's visualize vehicles data
# How does missing data show up?
plt.figure(figsize=(10,3))
plt.title('Vehicles')
plt.plot(df['Vehicles'], label='target')
plt.xticks(fontsize=14, rotation=45)
plt.legend()
plt.grid()
```



In [92]:
```
#DWB#

plt.figure(figsize=(10,3))
plt.title('Average Speed (mph), but forget you have seen it')
plt.plot(df['Average Speed (mph)'], label='speed')
```

```
plt.xticks(fontsize=14, rotation=45)
plt.legend()
plt.grid()
```

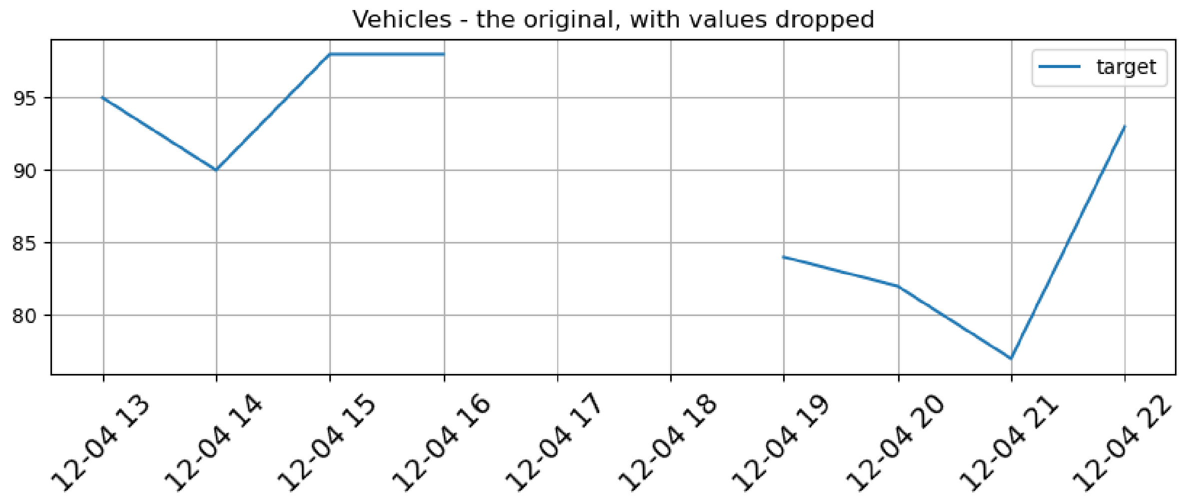


In [93]:
```
#DWB#

plt.figure(figsize=(10,3))
plt.title('Accidents, but forget you have seen it')
plt.plot(df['Accidents'], label='accidents')
plt.xticks(fontsize=14, rotation=45)
plt.legend()
plt.grid()
```

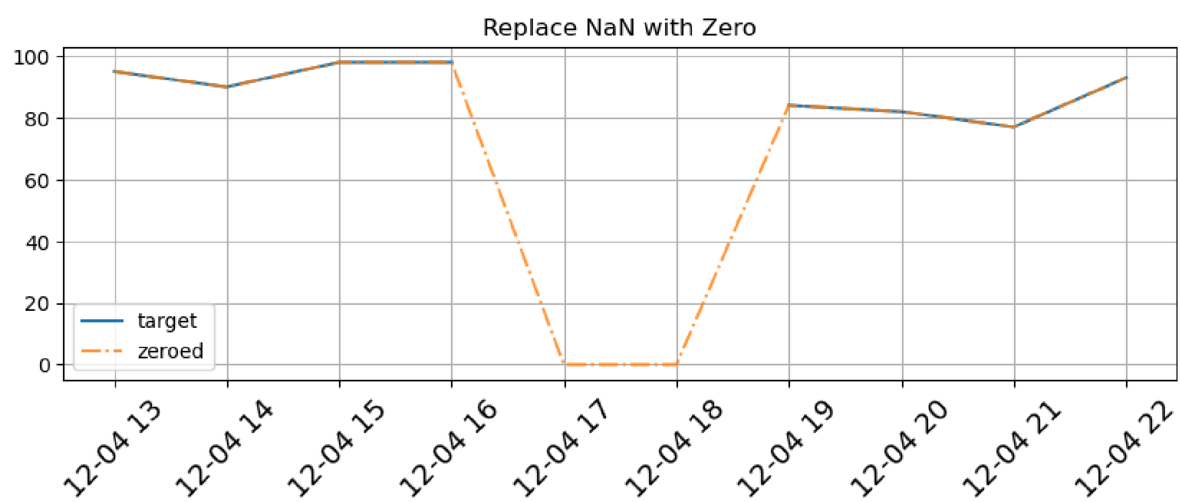
DWB# Let's see the Vehicles, again, since that is what we'll mostly be messing with.

In [94]:
```
#DWB#

plt.figure(figsize=(10,3))

plt.title('Vehicles - the original, with values dropped')
```

```
plt.plot(df['Vehicles'], label='target')
plt.xticks(fontsize=14, rotation=45)
plt.legend()
plt.grid()
```

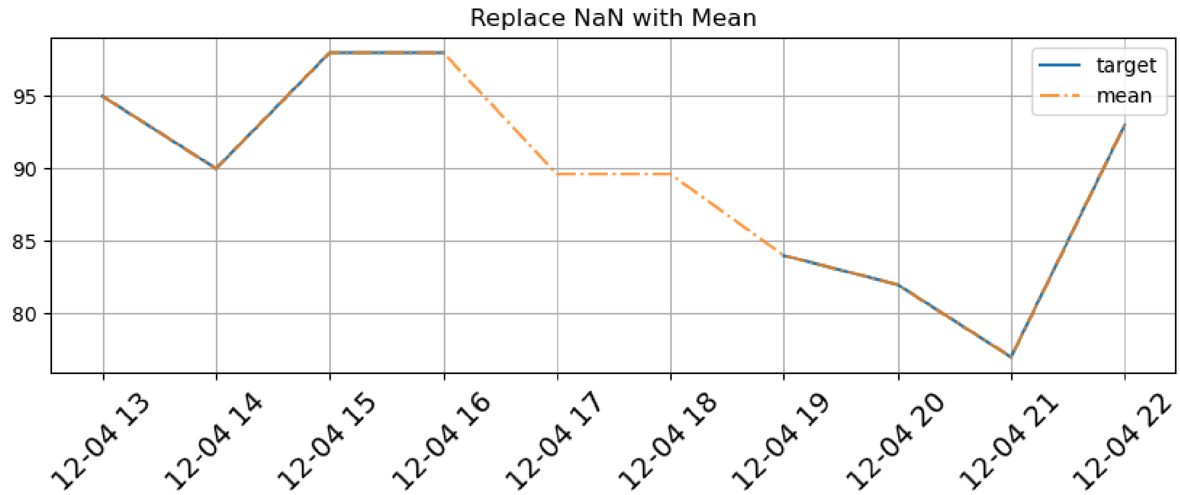


In [95]:
```
# Replace missing values with zero

plt.figure(figsize=(10,3))
plt.title('Replace NaN with Zero')
plt.plot(df['Vehicles'], label='target')

# fillna to replace NaNs with provided value
vehicles = df['Vehicles'].fillna(0)

plt.plot(vehicles,ls='-.',alpha=0.8,label='zeroed')
plt.xticks(fontsize=14, rotation=45)
plt.legend()
plt.grid()
```

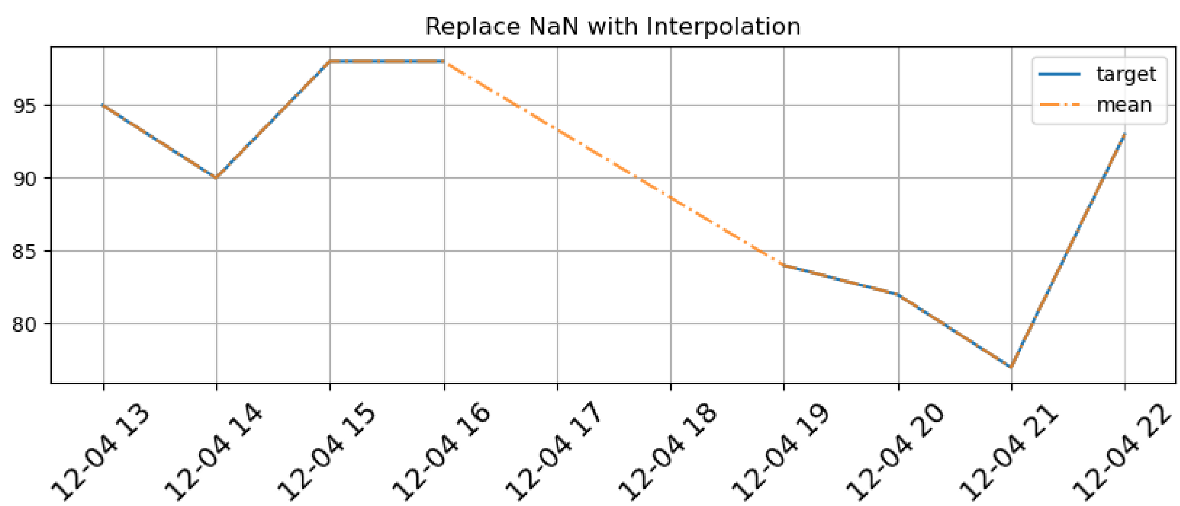


In [96]:
```
# Replace missing values with mean value for that attribute
plt.figure(figsize=(10,3))
plt.title('Replace NaN with Mean')
plt.plot(df['Vehicles'], label='target')
```

```
# fillna to replace NaNs with provided value
vehicles = df['Vehicles'].fillna(df['Vehicles'].mean())

plt.plot(vehicles,ls='-.',alpha=0.8,label='mean')
plt.xticks(fontsize=14, rotation=45)
plt.legend()
plt.grid()
```



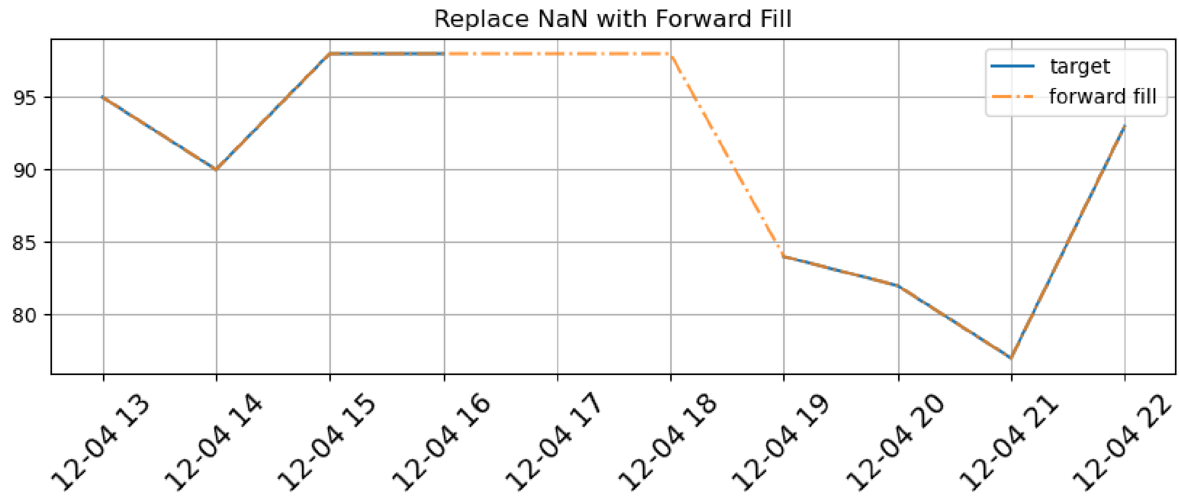
In [97]:
```
# Replace missing values with interpolated value for that attribute
plt.figure(figsize=(10,3))
plt.title('Replace NaN with Interpolation')
plt.plot(df['Vehicles'], label='target')

vehicles = df['Vehicles'].interpolate()

plt.plot(vehicles,ls='-.',alpha=0.8,label='mean')
plt.xticks(fontsize=14, rotation=45)
plt.legend()
plt.grid()
```



In [98]:
```
vehicles
```

Out[98]:
```
TimeStamp
2018-12-04 13:00:00    95.000000
2018-12-04 14:00:00    90.000000
2018-12-04 15:00:00    98.000000
2018-12-04 16:00:00    98.000000
2018-12-04 17:00:00    93.333333
2018-12-04 18:00:00    88.666667
2018-12-04 19:00:00    84.000000
2018-12-04 20:00:00    82.000000
2018-12-04 21:00:00    77.000000
2018-12-04 22:00:00    93.000000
Name: Vehicles, dtype: float64
```

In [99]:
```
# Replace missing values with previous valid value for that attribute
plt.figure(figsize=(10,3))
plt.title('Replace NaN with Forward Fill')
plt.plot(df['Vehicles'], label='target')

vehicles = df['Vehicles'].fillna(method='ffill')

plt.plot(vehicles,ls='-.',alpha=0.8,label='forward fill')
plt.xticks(fontsize=14, rotation=45)
plt.legend()
plt.grid()
```

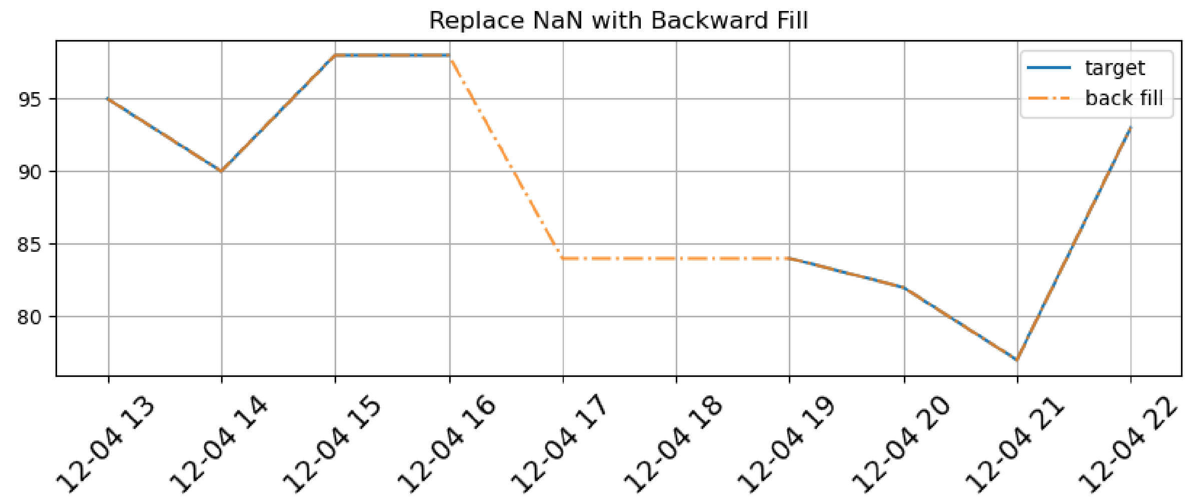


In [100…
```
vehicles
```

Out[100]:
```
TimeStamp
2018-12-04 13:00:00    95.0
2018-12-04 14:00:00    90.0
2018-12-04 15:00:00    98.0
2018-12-04 16:00:00    98.0
2018-12-04 17:00:00    98.0
2018-12-04 18:00:00    98.0
2018-12-04 19:00:00    84.0
2018-12-04 20:00:00    82.0
2018-12-04 21:00:00    77.0
2018-12-04 22:00:00    93.0
Name: Vehicles, dtype: float64
```

In [101…

```
# Replace missing values with next valid value for that attribute
plt.figure(figsize=(10,3))
plt.title('Replace NaN with Backward Fill')
plt.plot(df['Vehicles'], label='target')

vehicles = df['Vehicles'].fillna(method='bfill')

plt.plot(vehicles,ls='-.',alpha=0.8,label='back fill')
plt.xticks(fontsize=14, rotation=45)
plt.legend()
plt.grid()
```



In [102…

```
vehicles
```

Out[102]:

```
TimeStamp
2018-12-04 13:00:00    95.0
2018-12-04 14:00:00    90.0
2018-12-04 15:00:00    98.0
2018-12-04 16:00:00    98.0
2018-12-04 17:00:00    84.0
2018-12-04 18:00:00    84.0
2018-12-04 19:00:00    84.0
2018-12-04 20:00:00    82.0
2018-12-04 21:00:00    77.0
2018-12-04 22:00:00    93.0
Name: Vehicles, dtype: float64
```

In [103…

```
df
```

Out[103]:

TimeStamp	Vehicles	Average Speed (mph)	Accidents
2018-12-04 13:00:00	95.0	38.0	0.0
2018-12-04 14:00:00	90.0	32.0	1.0
2018-12-04 15:00:00	98.0	30.0	1.0
2018-12-04 16:00:00	98.0	26.0	3.0
2018-12-04 17:00:00	NaN	NaN	NaN
2018-12-04 18:00:00	NaN	NaN	NaN
2018-12-04 19:00:00	84.0	35.0	2.0
2018-12-04 20:00:00	82.0	40.0	0.0
2018-12-04 21:00:00	77.0	45.0	0.0
2018-12-04 22:00:00	93.0	45.0	1.0

In [104…

```
# Now that we know different ways of handling missing values
# Let's pick an appropriate scheme for replacing missing values

# Vehicles and Average Speed...interpolate
df['Vehicles'] = df['Vehicles'].interpolate()
df['Average Speed (mph)'] = df['Average Speed (mph)'].interpolate()
# Accidents...interpolate or use mean values
df['Accidents'] = df['Accidents'].fillna(df['Accidents'].mean())
```

In [105…

```
df
```

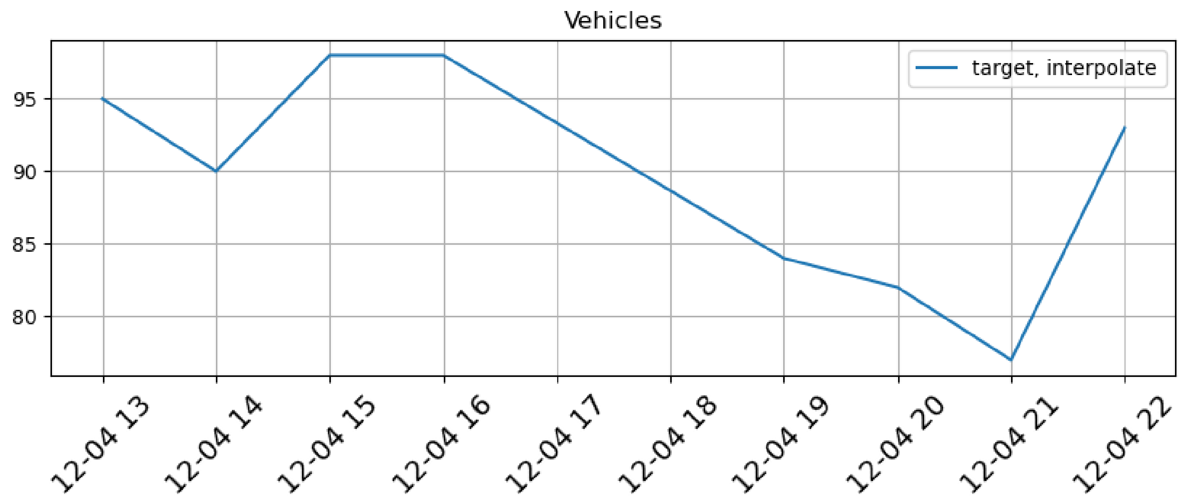
Out[105]:

TimeStamp	Vehicles	Average Speed (mph)	Accidents
2018-12-04 13:00:00	95.000000	38.0	0.0
2018-12-04 14:00:00	90.000000	32.0	1.0
2018-12-04 15:00:00	98.000000	30.0	1.0
2018-12-04 16:00:00	98.000000	26.0	3.0
2018-12-04 17:00:00	93.333333	29.0	1.0
2018-12-04 18:00:00	88.666667	32.0	1.0
2018-12-04 19:00:00	84.000000	35.0	2.0
2018-12-04 20:00:00	82.000000	40.0	0.0
2018-12-04 21:00:00	77.000000	45.0	0.0
2018-12-04 22:00:00	93.000000	45.0	1.0

In [106…

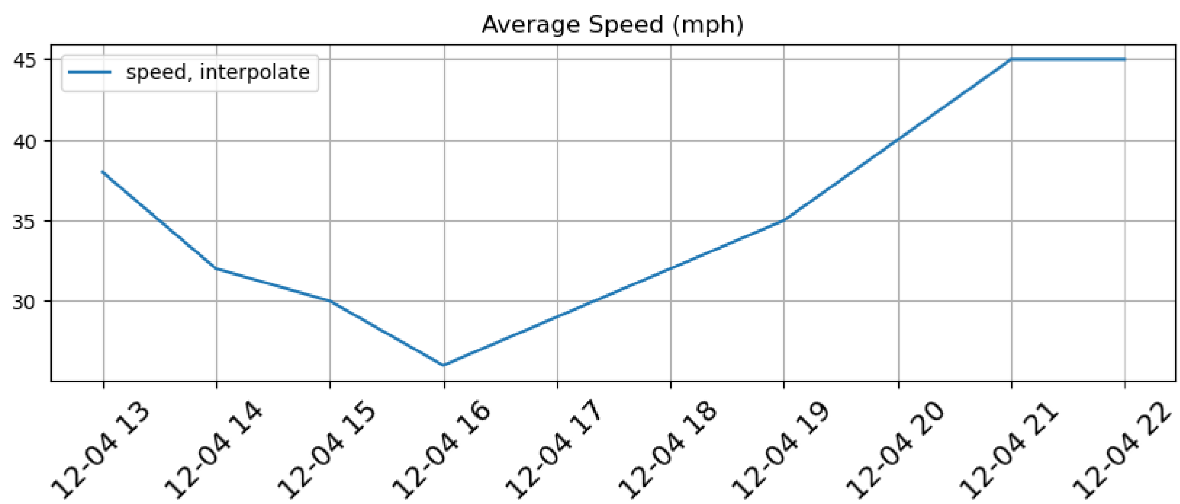
```
#DWB#
```

```
plt.figure(figsize=(10,3))
plt.title('Vehicles')
plt.plot(df['Vehicles'], label='target, interpolate')
plt.xticks(fontsize=14, rotation=45)
plt.legend()
plt.grid()
```



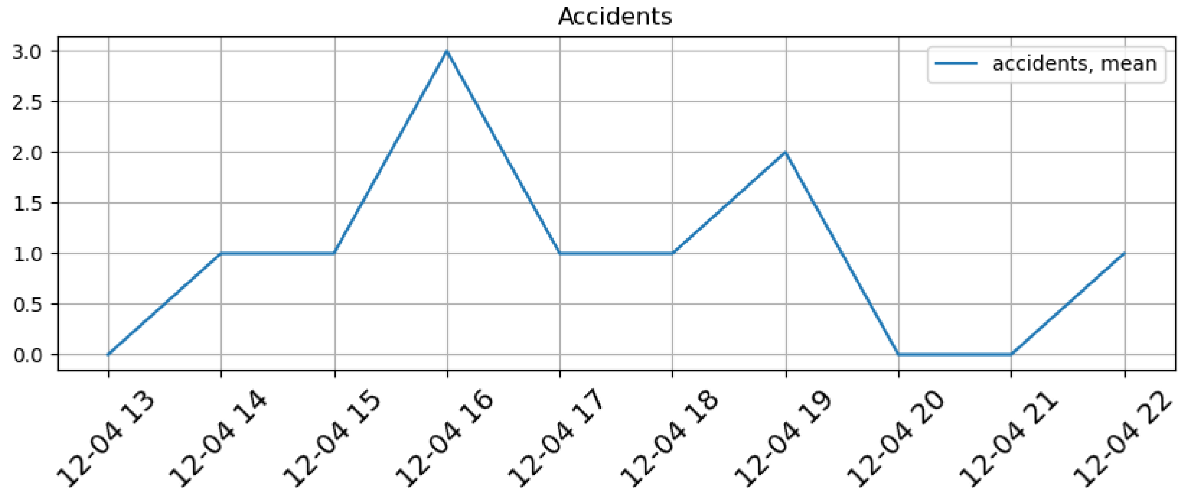
In [107…

```
#DWB#

plt.figure(figsize=(10,3))
plt.title('Average Speed (mph)')
plt.plot(df['Average Speed (mph)'], label='speed, interpolate')
plt.xticks(fontsize=14, rotation=45)
plt.legend()
plt.grid()
```



In [108…

```
#DWB#

plt.figure(figsize=(10,3))
plt.title('Accidents')
plt.plot(df['Accidents'], label='accidents, mean')
plt.xticks(fontsize=14, rotation=45)
plt.legend()
plt.grid()
```



Independent Data

In [109…

```
# Example of data that is not time dependent
# Each row is independent
df = pd.read_csv('VehicleTrafficRoads.csv', index_col=0)
```

In [142…

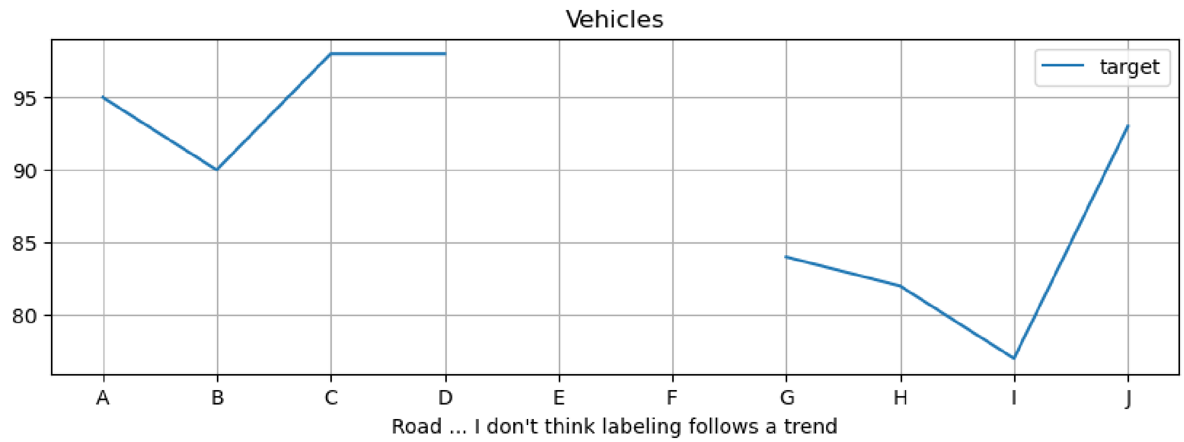
```
df
```

Out[142]:

Road	Vehicles	Average Speed (mph)	Accidents
A	95.0	38.0	0.0
B	90.0	32.0	1.0
C	98.0	30.0	1.0
D	98.0	26.0	3.0
E	NaN	NaN	NaN
F	NaN	NaN	NaN
G	84.0	35.0	2.0
H	82.0	40.0	0.0
I	77.0	45.0	0.0
J	93.0	45.0	1.0

In [143…

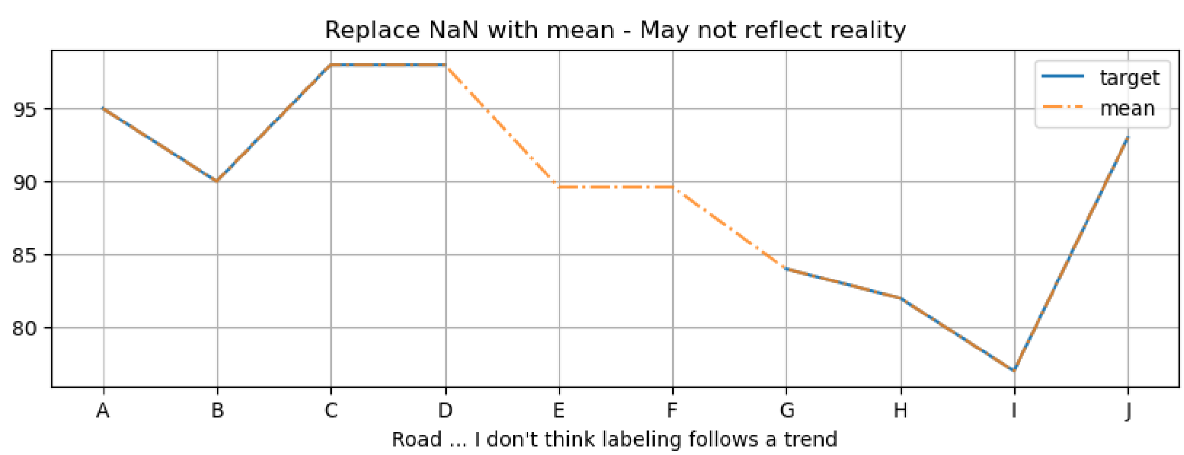
```
#DWB#
plt.figure(figsize=(10,3))
plt.title('Vehicles')
plt.plot(df['Vehicles'], label='target')
plt.xlabel("Road ... I don't think labeling follows a trend")
plt.legend()
plt.grid()
```



In [30]:
```
df.mean()
```

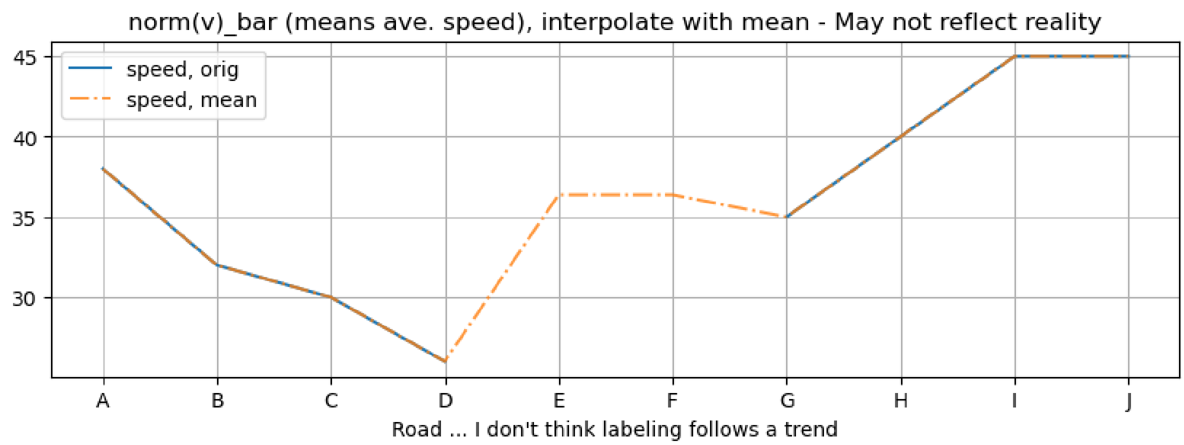
Out[30]:
```
Vehicles               89.625
Average Speed (mph)    36.375
Accidents               1.000
dtype: float64
```

In [131…
```
# Substitute computed average of other rows
# In this case, Rows E and F look identical
# Data stored for Road E and F may not reflect reality

#begin:  DWB1#
plt.figure(figsize=(10,3))
plt.title('Replace NaN with mean - May not reflect reality')
plt.plot(df['Vehicles'], label='target')
#endof:  DWB1#

df.fillna(df.mean())

#begin:  DWB2#
maynotreflectreality = df.fillna(df.mean())

plt.plot(maynotreflectreality['Vehicles'],ls='-.',alpha=0.8,label='mean')
plt.xlabel("Road ... I don't think labeling follows a trend")
plt.legend()
plt.grid()
#endof:  DWB2#
```



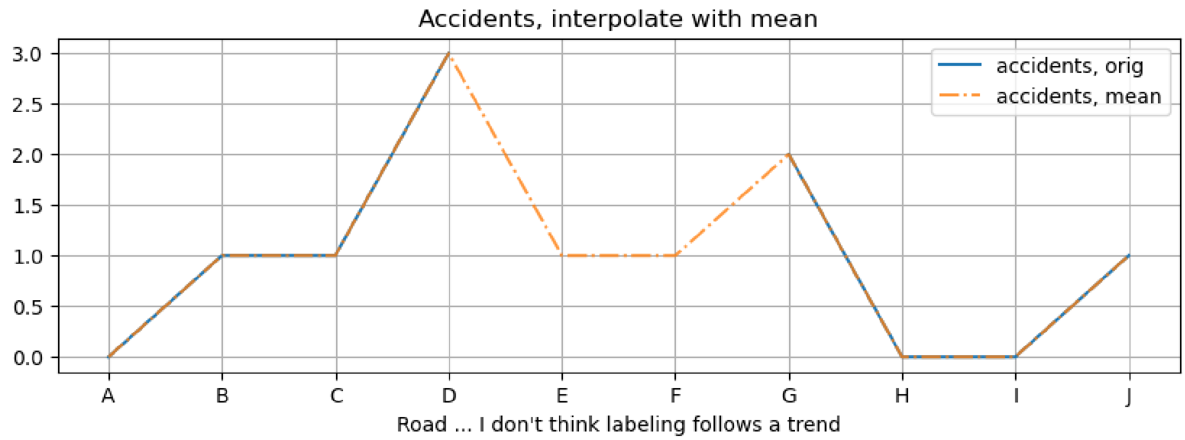
In [140…

```
#DWB3#

df = pd.read_csv('VehicleTrafficRoads.csv', index_col=0)

plt.figure(figsize=(10,3))
plt.title('norm(v)_bar (means ave. speed), interpolate with mean - May not reflect
plt.plot(df['Average Speed (mph)'], label='speed, orig')

maynotreflectreality = df.fillna(df.mean())

plt.plot(maynotreflectreality['Average Speed (mph)'],ls='-.',alpha=0.8,label='speed
plt.xlabel("Road ... I don't think labeling follows a trend")
plt.legend()
plt.grid()
```



In [145…

```
#DWB4#

df = pd.read_csv('VehicleTrafficRoads.csv', index_col=0)

plt.figure(figsize=(10,3))
plt.title('Accidents, interpolate with mean')
plt.plot(df['Accidents'], label='accidents, orig')

maynotreflectreality = df.fillna(df.mean())

plt.plot(maynotreflectreality['Accidents'],ls='-.',alpha=0.8,label='accidents, mean
plt.xlabel("Road ... I don't think labeling follows a trend")
plt.legend()
plt.grid()
```



In [146…

```
# Better option here is to simply drop NA rows
# how = all Drop if all columns are NA
# how = any Drop if any one of the columns contain NA
df.dropna(how='all',inplace=True)
```

In [147…

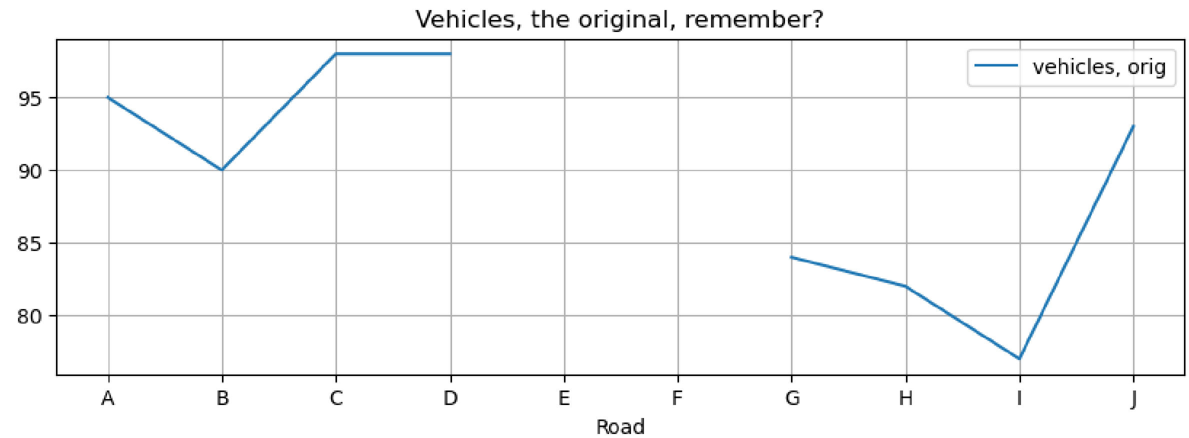
```
df
```

Out[147]:

Road	Vehicles	Average Speed (mph)	Accidents
A	95.0	38.0	0.0
B	90.0	32.0	1.0
C	98.0	30.0	1.0
D	98.0	26.0	3.0
G	84.0	35.0	2.0
H	82.0	40.0	0.0
I	77.0	45.0	0.0
J	93.0	45.0	1.0

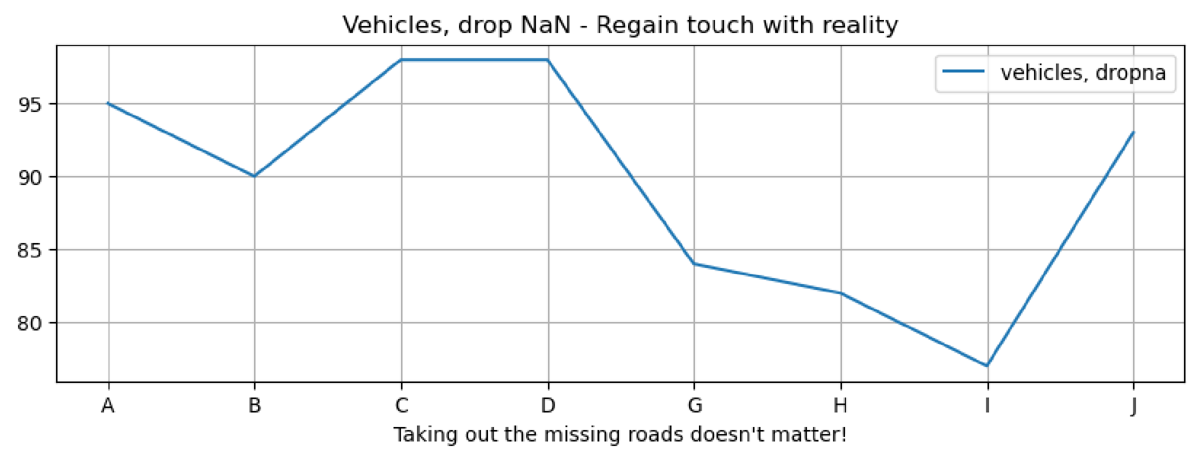
In [148…

```
#DWB#
df = pd.read_csv('VehicleTrafficRoads.csv', index_col=0)

plt.figure(figsize=(10,3))
plt.title('Vehicles, the original, remember?')
plt.plot(df['Vehicles'], label='vehicles, orig')
plt.xlabel("Road")
plt.legend()
plt.grid()
```



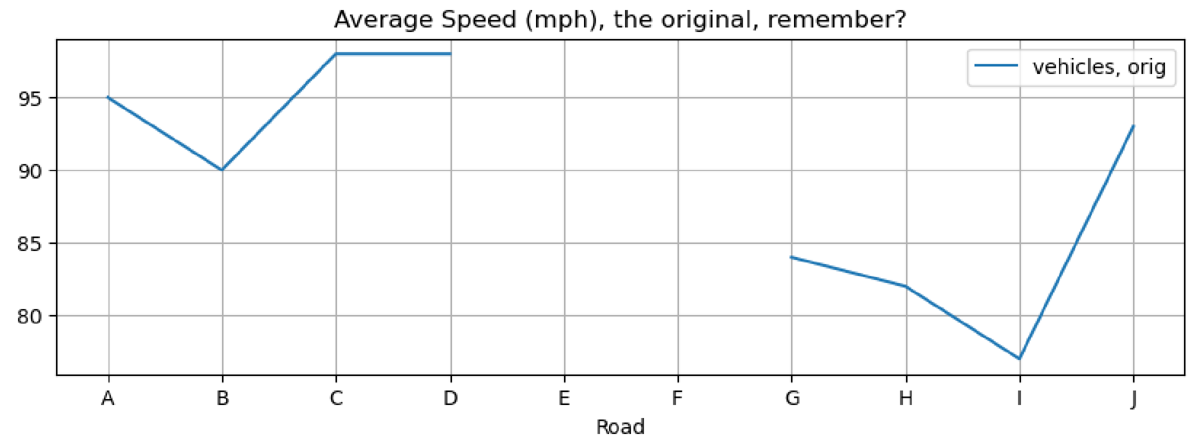
In [149…

```
#DWB#
df = pd.read_csv('VehicleTrafficRoads.csv', index_col=0)

plt.figure(figsize=(10,3))
plt.title('Vehicles, drop NaN - Regain touch with reality')

df.dropna(how='all', inplace=True)

plt.plot(df['Vehicles'], label='vehicles, dropna')
plt.xlabel("Taking out the missing roads doesn't matter!")
plt.legend()
plt.grid()
```



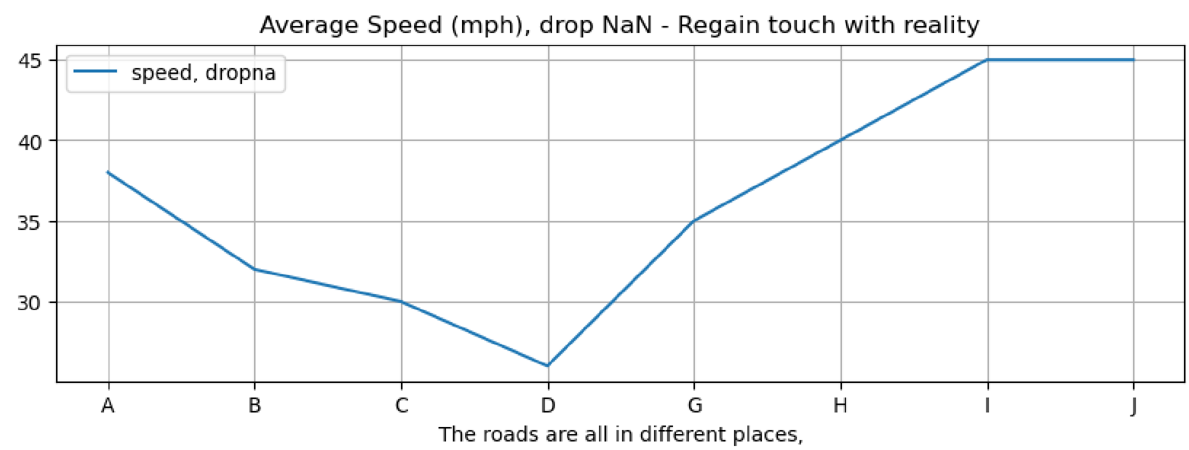
In [150…

```
#DWB#
df = pd.read_csv('VehicleTrafficRoads.csv', index_col=0)

plt.figure(figsize=(10,3))
plt.title('Average Speed (mph), the original, remember?')
plt.plot(df['Vehicles'], label='vehicles, orig')
plt.xlabel("Road")
plt.legend()
plt.grid()
```



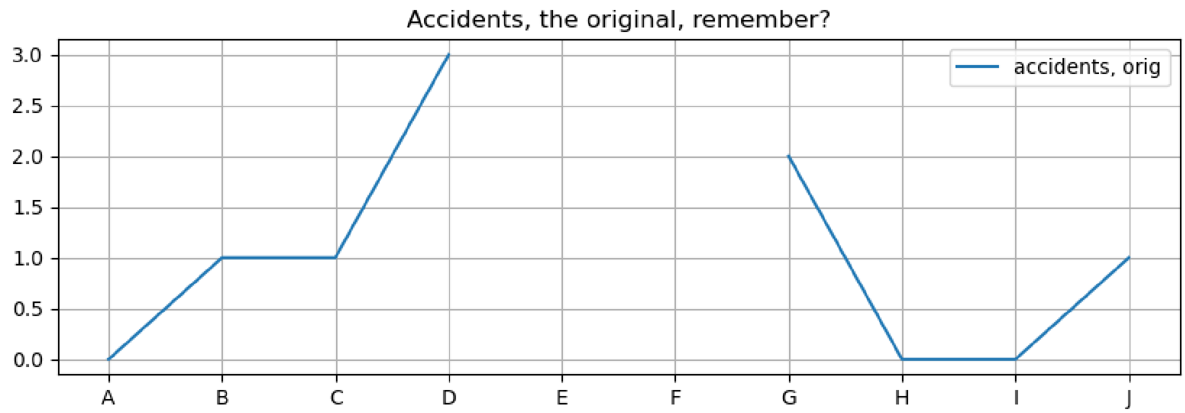
In [151…

```
#DWB#
df = pd.read_csv('VehicleTrafficRoads.csv', index_col=0)

plt.figure(figsize=(10,3))
plt.title('Average Speed (mph), drop NaN - Regain touch with reality')

df.dropna(how='all', inplace=True)

plt.plot(df['Average Speed (mph)'], label='speed, dropna')
plt.xlabel("The roads are all in different places,")
plt.legend()
plt.grid()
```



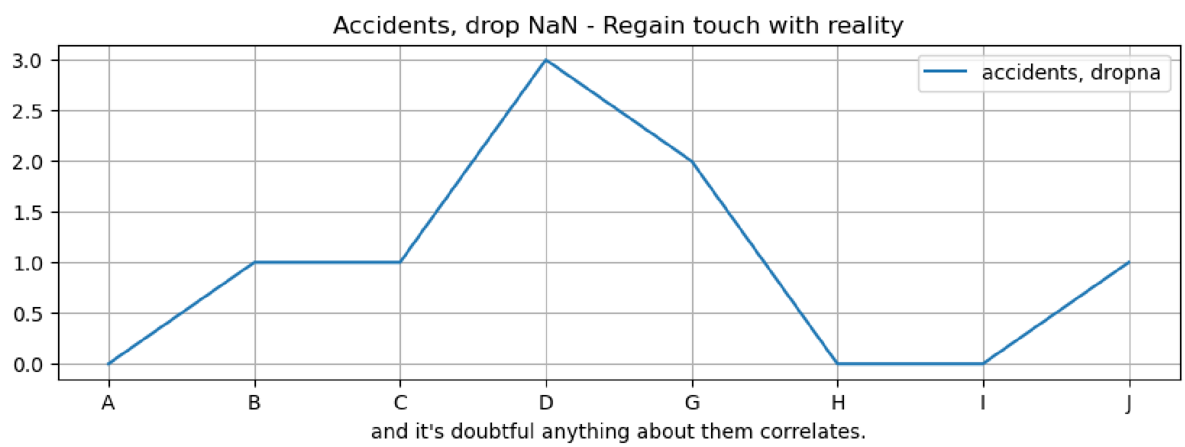
In [152…

```
#DWB#
df = pd.read_csv('VehicleTrafficRoads.csv', index_col=0)

plt.figure(figsize=(10,3))
plt.title('Accidents, the original, remember?')
plt.plot(df['Accidents'], label='accidents, orig')
plt.legend()
plt.grid()
```



In [153…

```
#DWB#
df = pd.read_csv('VehicleTrafficRoads.csv', index_col=0)

plt.figure(figsize=(10,3))
plt.title('Accidents, drop NaN - Regain touch with reality')

df.dropna(how='all', inplace=True)

plt.plot(df['Accidents'], label='accidents, dropna')
plt.xlabel("and it's doubtful anything about them correlates.")
plt.legend()
plt.grid()
```



Impute Missing Values from Similar Data

In [154…

```
# Some instances have missing features
# There are three types of plants: Iris-setosa, Iris-virginica, Iris-versicolor
# In this case, we can find mean value of an attribute for each type of plant
# and use it to substitute the missing values
df = pd.read_csv('IrisMissingData.csv')
```

In [155…

```
df
```

Out[155]:

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	NaN	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [156…

```
# Look for any columns that have NA
df.isna().any(axis=0)
```

Out[156]:

```
sepal_length     True
sepal_width      True
petal_length     True
petal_width      True
class           False
dtype: bool
```

In [158…

```
# Look for any rows that have NA
rows_missing_values = df.isna().any(axis=1)
```

In [159…

```
df[rows_missing_values]
```

Out[159]:

	sepal_length	sepal_width	petal_length	petal_width	class
6	4.6	NaN	1.4	0.3	Iris-setosa
7	5.0	3.4	NaN	0.2	Iris-setosa
12	4.8	3.0	1.4	NaN	Iris-setosa
62	NaN	2.2	4.0	1.0	Iris-versicolor
64	5.6	2.9	3.6	NaN	Iris-versicolor
80	5.5	NaN	NaN	1.1	Iris-versicolor
127	6.1	NaN	4.9	1.8	Iris-virginica
128	6.4	2.8	NaN	2.1	Iris-virginica
140	6.7	3.1	NaN	2.4	Iris-virginica
145	NaN	3.0	5.2	2.3	Iris-virginica

In [160…

```
# Find Summary Statistics for Each Class
# Impute values based on class
# https://stackoverflow.com/questions/19966018/pandas-filling-missing-values-by-mea
group_class = df.groupby('class')
```

In [161…

```
# First few rows of each group
group_class.head(2)
```

Out[161]:

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
50	7.0	3.2	4.7	1.4	Iris-versicolor
51	6.4	3.2	4.5	1.5	Iris-versicolor
100	6.3	3.3	6.0	2.5	Iris-virginica
101	5.8	2.7	5.1	1.9	Iris-virginica

In [164…

```
# Attribute Mean value is different for each group
group_class.mean()
```

Out[164]:

	sepal_length	sepal_width	petal_length	petal_width
class				
Iris-setosa	5.006000	3.418367	1.463265	0.246939
Iris-versicolor	5.934694	2.777551	4.269388	1.326531
Iris-virginica	6.585714	2.973469	5.550000	2.026000

In [168…

```
# Compared to mean value for entire dataset
df.mean()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/nanops.py:1
680, in _ensure_numeric(x)
   1679 try:
-> 1680     x = x.astype(np.complex128)
   1681 except (TypeError, ValueError):

ValueError: complex() arg is a malformed string

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/nanops.py:1
683, in _ensure_numeric(x)
   1682 try:
-> 1683     x = x.astype(np.float64)
   1684 except ValueError as err:
   1685     # GH#29941 we get here with object arrays containing strs

ValueError: could not convert string to float: 'Iris-setosaIris-setosaIris-setosaI
ris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-s
etosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosa
Iris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-
setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setos
aIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris
-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-seto
saIris-setosaIris-setosaIris-versicolorIris-versicolorIris-versicolorIris-versicol
orIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-
versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versico
lorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris
-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versic
olorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIri
s-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versi
colorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIr
is-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-vers
icolorIris-versicolorIris-versicolorIris-virginicaIris-virginicaIris-virginicaIris
-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIr
is-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginica
Iris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virgini
caIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virgi
nicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-vir
ginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-v
irginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris
-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginica'

The above exception was the direct cause of the following exception:

TypeError                                 Traceback (most recent call last)
Cell In[168], line 2
      1 # Compared to mean value for entire dataset
----> 2 df.mean()

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/generic.py:
11556, in NDFrame._add_numeric_operations.<locals>.mean(self, axis, skipna, numeri
c_only, **kwargs)
```

```
  11539 @doc(
  11540     _num_doc,
  11541     desc="Return the mean of the values over the requested axis.",
   (...)
  11554     **kwargs,
  11555 ):
> 11556     return NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/generic.py:
11201, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  11194 def mean(
  11195     self,
  11196     axis: Axis | None = 0,
   (...)
  11199     **kwargs,
  11200 ) -> Series | float:
> 11201     return self._stat_function(
  11202         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
  11203     )

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/generic.py:
11158, in NDFrame._stat_function(self, name, func, axis, skipna, numeric_only, **k
wargs)
  11154     nv.validate_stat_func((), kwargs, fname=name)
  11156 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 11158 return self._reduce(
  11159     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_only
  11160 )

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/frame.py:10
524, in DataFrame._reduce(self, op, name, axis, skipna, numeric_only, filter_type,
**kwds)
  10520     df = df.T
  10522 # After possibly _get_data and transposing, we are now in the
  10523 #  simple case where we can use BlockManager.reduce
> 10524 res = df._mgr.reduce(blk_func)
  10525 out = df._constructor(res).iloc[0]
  10526 if out_dtype is not None:

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/internals/m
anagers.py:1534, in BlockManager.reduce(self, func)
   1532 res_blocks: list[Block] = []
   1533 for blk in self.blocks:
-> 1534     nbs = blk.reduce(func)
   1535     res_blocks.extend(nbs)
   1537 index = Index([None])  # placeholder

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/internals/b
locks.py:339, in Block.reduce(self, func)
    333 @final
    334 def reduce(self, func) -> list[Block]:
    335     # We will apply the function and reshape the result into a single-row
    336     #  Block with the same mgr_locs; squeezing will be done at a higher le
vel
    337     assert self.ndim == 2
--> 339     result = func(self.values)
```

```
    341     if self.values.ndim == 1:
    342         # TODO(EA2D): special case not needed with 2D EAs
    343         res_values = np.array([[result]])

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/frame.py:10
487, in DataFrame._reduce.<locals>.blk_func(values, axis)
  10485     return values._reduce(name, skipna=skipna, **kwds)
  10486 else:
> 10487     return op(values, axis=axis, skipna=skipna, **kwds)

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/nanops.py:9
6, in disallow.__call__.<locals>._f(*args, **kwargs)
     94 try:
     95     with np.errstate(invalid="ignore"):
---> 96         return f(*args, **kwargs)
     97 except ValueError as e:
     98     # we want to transform an object array
     99     # ValueError message to the more typical TypeError
    100     # e.g. this is normally a disallowed function on
    101     # object arrays that contain strings
    102     if is_object_dtype(args[0]):

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/nanops.py:1
58, in bottleneck_switch.__call__.<locals>.f(values, axis, skipna, **kwds)
    156         result = alt(values, axis=axis, skipna=skipna, **kwds)
    157 else:
--> 158     result = alt(values, axis=axis, skipna=skipna, **kwds)
    160 return result

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/nanops.py:4
21, in _datetimelike_compat.<locals>.new_func(values, axis, skipna, mask, **kwarg
s)
    418 if datetimelike and mask is None:
    419     mask = isna(values)
--> 421 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
    423 if datetimelike:
    424     result = _wrap_results(result, orig_values.dtype, fill_value=iNaT)

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/nanops.py:7
27, in nanmean(values, axis, skipna, mask)
    724     dtype_count = dtype
    726 count = _get_counts(values.shape, mask, axis, dtype=dtype_count)
--> 727 the_sum = _ensure_numeric(values.sum(axis, dtype=dtype_sum))
    729 if axis is not None and getattr(the_sum, "ndim", False):
    730     count = cast(np.ndarray, count)

File ~/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/nanops.py:1
686, in _ensure_numeric(x)
   1683         x = x.astype(np.float64)
   1684     except ValueError as err:
   1685         # GH#29941 we get here with object arrays containing strs
-> 1686         raise TypeError(f"Could not convert {x} to numeric") from err
   1687 else:
   1688     if not np.any(np.imag(x)):

TypeError: Could not convert ['Iris-setosaIris-setosaIris-setosaIris-setosaIris-se
```

```
tosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaI
ris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-s
etosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosa
Iris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-
setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setos
aIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris-setosaIris
-setosaIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolor
Iris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-ve
rsicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolo
rIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-v
ersicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicol
orIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-
versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versico
lorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris
-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versicolorIris-versic
olorIris-versicolorIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-vi
rginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-
virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIri
s-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaI
ris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginic
aIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virgin
icaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virg
inicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-virginicaIris-vi
rginicaIris-virginicaIris-virginicaIris-virginicaIris-virginica'] to numeric
```

In [169…

```
#DWB# Try to make it work
df.mean(numeric_only=True)
```

Out[169]:

```
sepal_length    5.836486
sepal_width     3.056463
petal_length    3.748630
petal_width     1.205405
dtype: float64
```

In [170…

```
# For each group, use group level averages to fill missing values
df['sepal_length'] = group_class['sepal_length'].transform(lambda x: x.fillna(x.mea
df['sepal_width'] = group_class['sepal_width'].transform(lambda x: x.fillna(x.mean(
df['petal_length'] = group_class['petal_length'].transform(lambda x: x.fillna(x.mea
df['petal_width'] = group_class['petal_width'].transform(lambda x: x.fillna(x.mean(
```

In [171…

```
# Let's now check the rows that had missing values
df[rows_missing_values]
```

Out[171]:

	sepal_length	sepal_width	petal_length	petal_width	class
6	4.600000	3.418367	1.400000	0.300000	Iris-setosa
7	5.000000	3.400000	1.463265	0.200000	Iris-setosa
12	4.800000	3.000000	1.400000	0.246939	Iris-setosa
62	5.934694	2.200000	4.000000	1.000000	Iris-versicolor
64	5.600000	2.900000	3.600000	1.326531	Iris-versicolor
80	5.500000	2.777551	4.269388	1.100000	Iris-versicolor
127	6.100000	2.973469	4.900000	1.800000	Iris-virginica
128	6.400000	2.800000	5.550000	2.100000	Iris-virginica
140	6.700000	3.100000	5.550000	2.400000	Iris-virginica
145	6.585714	3.000000	5.200000	2.300000	Iris-virginica

In [ ]:

```

```