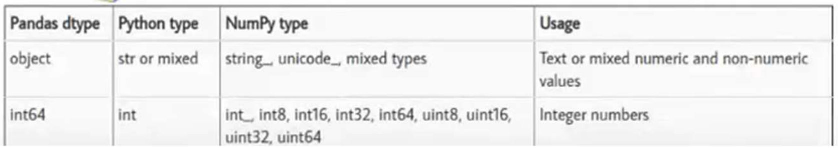
```python
import time

sns.set()
pal = sns.hls_palette(10, h=.5)
sns.set_palette(pal)

#Avoid display of scientific notation and show precision of 4 decimals:
pd.set_option('display.float_format', lambda x: '%.4f' % x)
```

```python
# Data Source
#Uber Trip Data from 9/1/2014 to 9/1/2015 with ~31M entries.

#Source: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

# Pull the data using the below command

#direct source: !curl -O https://s3.amazonaws.com/nyc-tlc/misc/uber_nyc_data.csv
df = pd.read_csv('uber_nyc_data.csv')
```

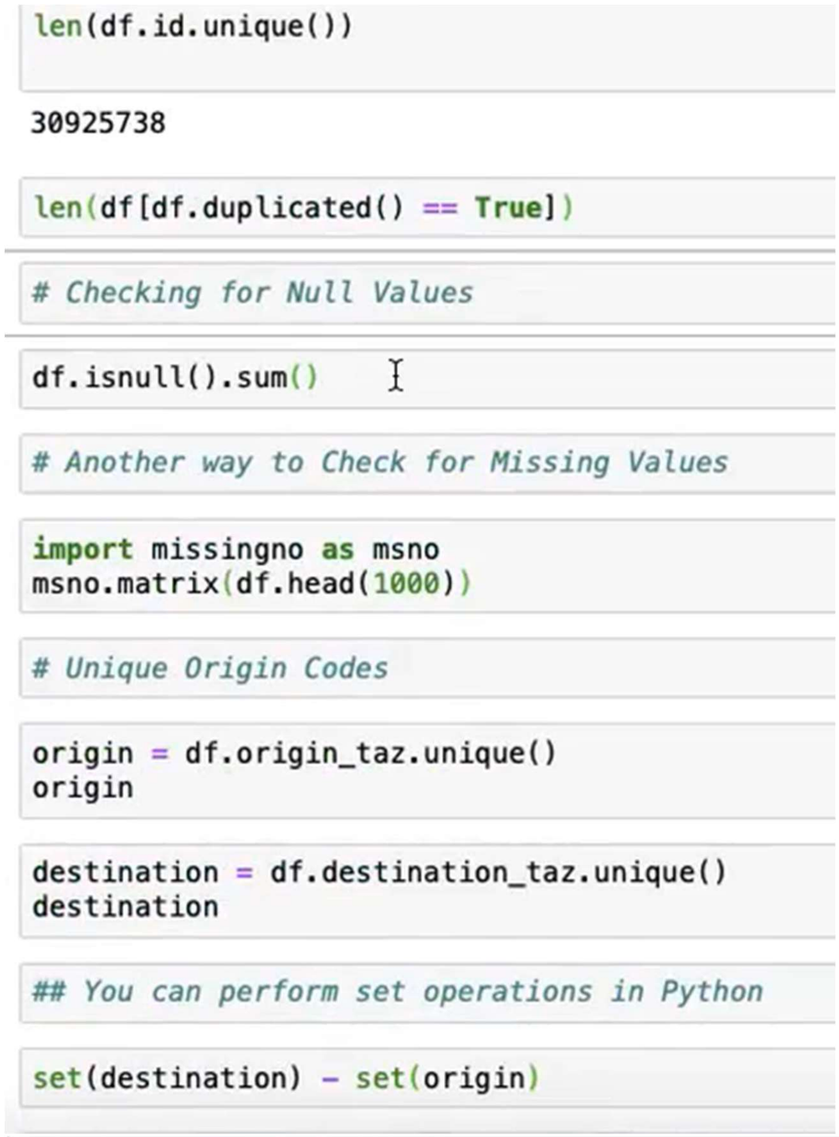
Objective

- To Clean the Data
- To Understand the Data
- To Generate Insights from Data

```python
df.info()
```



Pandas dtype	Python type	NumPy type	Usage
object	str or mixed	string_, unicode_, mixed types	Text or mixed numeric and non-numeric values
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64	Integer numbers



```
len(df.id.unique())
```

```
30925738
```

```
len(df[df.duplicated() == True])
```

```
# Checking for Null Values
```

```
df.isnull().sum()
```

```
# Another way to Check for Missing Values
```

```
import missingno as msno
msno.matrix(df.head(1000))
```

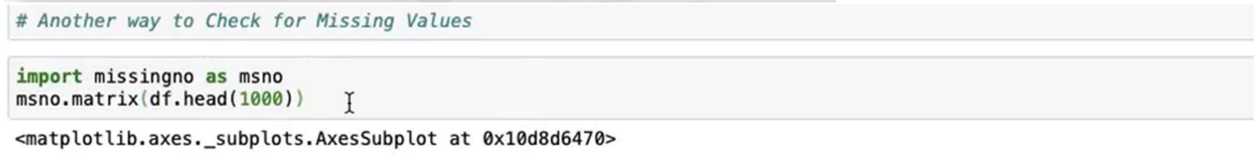
```
# Unique Origin Codes
```

```
origin = df.origin_taz.unique()
origin
```

```
destination = df.destination_taz.unique()
destination
```

```
## You can perform set operations in Python
```

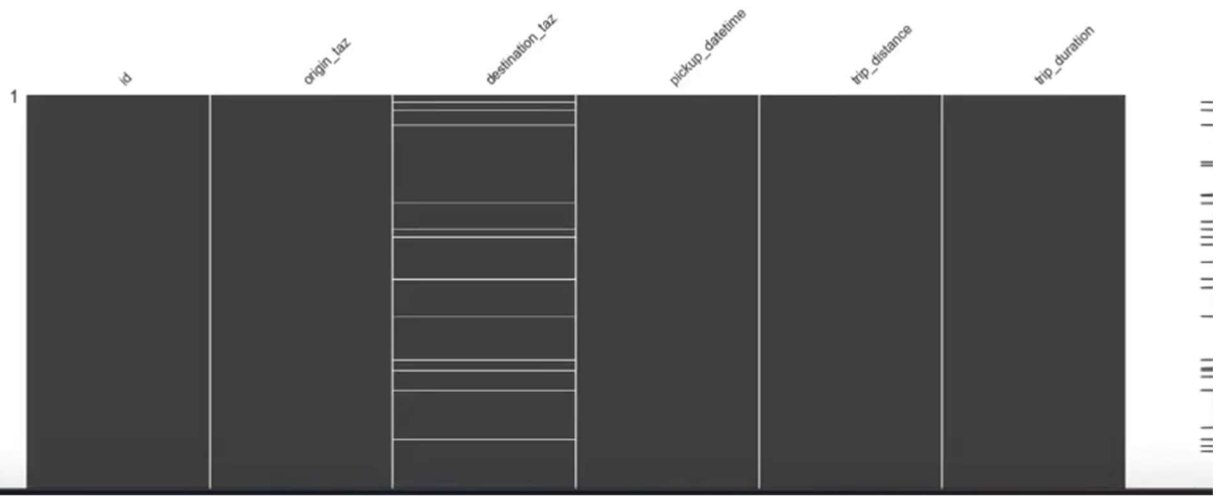
```
set(destination) - set(origin)
```

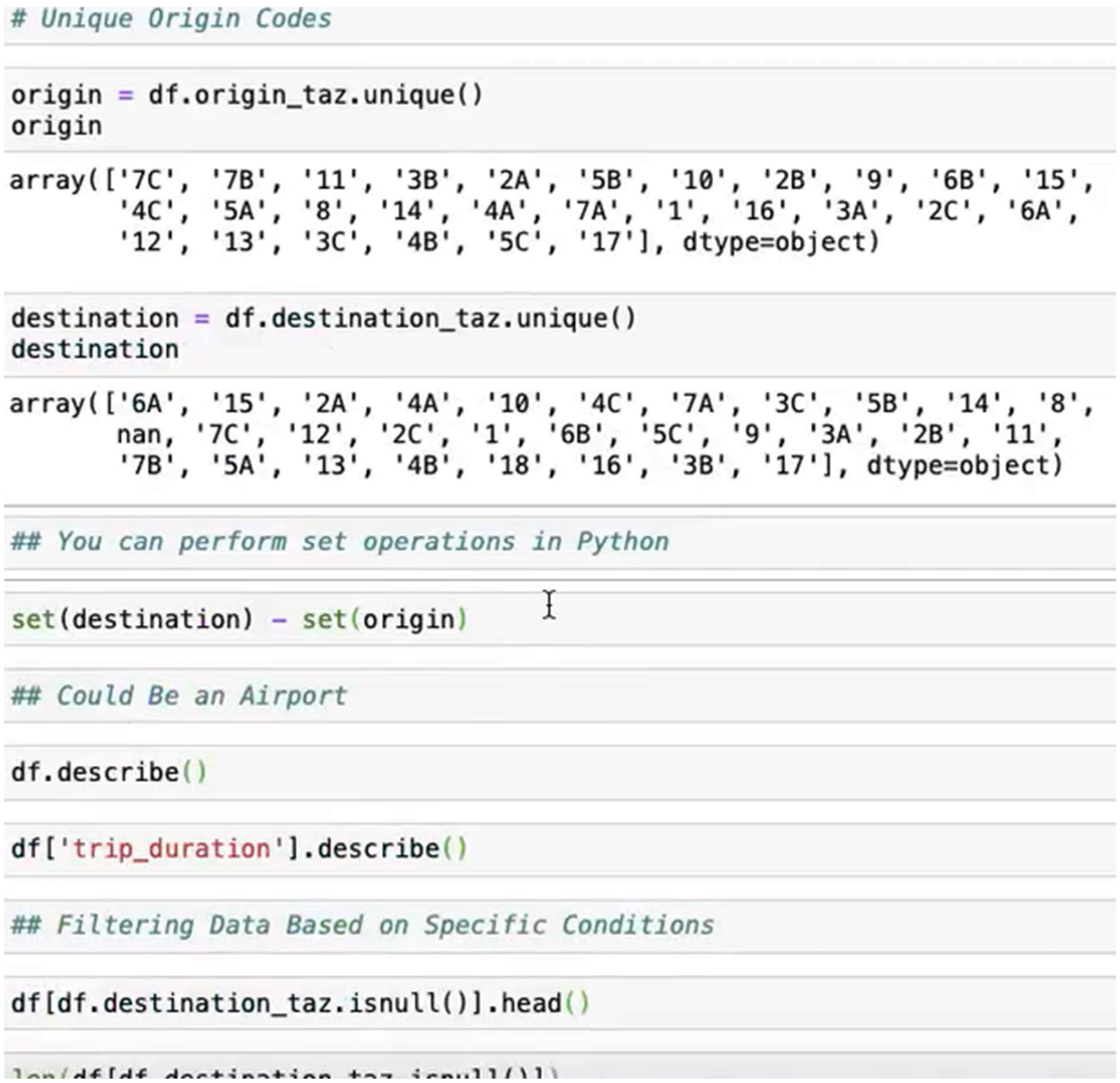


```
# Another way to Check for Missing Values
```

```
import missingno as msno
msno.matrix(df.head(1000))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10d8d6470>
```





```
# Unique Origin Codes
```

```
origin = df.origin_taz.unique()
origin
```

```
array(['7C', '7B', '11', '3B', '2A', '5B', '10', '2B', '9', '6B', '15',
       '4C', '5A', '8', '14', '4A', '7A', '1', '16', '3A', '2C', '6A',
       '12', '13', '3C', '4B', '5C', '17'], dtype=object)
```

```
destination = df.destination_taz.unique()
destination
```

```
array(['6A', '15', '2A', '4A', '10', '4C', '7A', '3C', '5B', '14', '8',
       nan, '7C', '12', '2C', '1', '6B', '5C', '9', '3A', '2B', '11',
       '7B', '5A', '13', '4B', '18', '16', '3B', '17'], dtype=object)
```

```
## You can perform set operations in Python
```

```
set(destination) - set(origin)
```

```
## Could Be an Airport
```

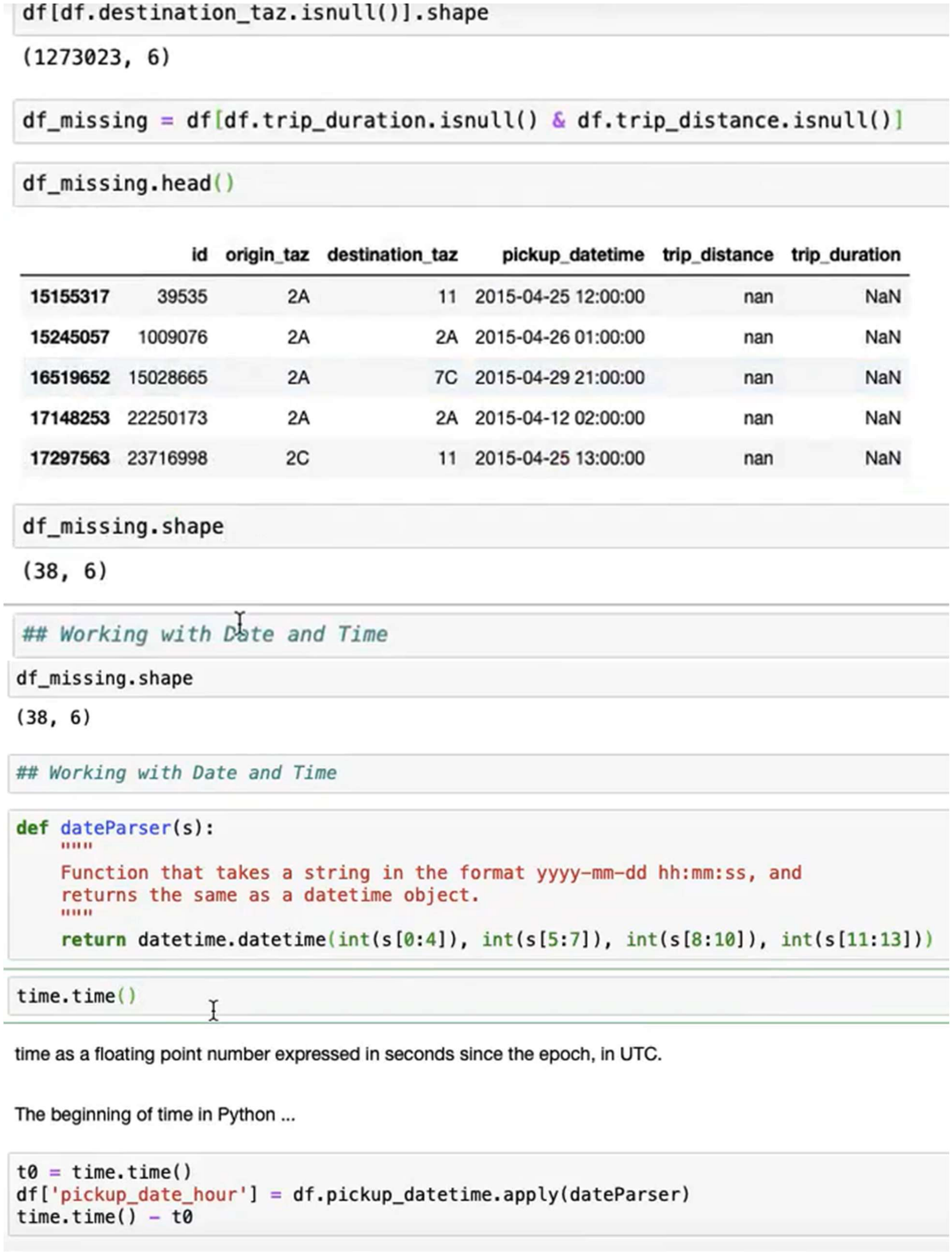
```
df.describe()
```

```
df['trip_duration'].describe()
```

```
## Filtering Data Based on Specific Conditions
```

```
df[df.destination_taz.isnull()].head()
```

```
len(df[df.destination_taz.isnull()])
```



```
df[df.destination_taz.isnull()].shape
```

```
(1273023, 6)
```

```
df_missing = df[df.trip_duration.isnull() & df.trip_distance.isnull()]
```

```
df_missing.head()
```

	id	origin_taz	destination_taz	pickup_datetime	trip_distance	trip_duration
15155317	39535	2A	11	2015-04-25 12:00:00	nan	NaN
15245057	1009076	2A	2A	2015-04-26 01:00:00	nan	NaN
16519652	15028665	2A	7C	2015-04-29 21:00:00	nan	NaN
17148253	22250173	2A	2A	2015-04-12 02:00:00	nan	NaN
17297563	23716998	2C	11	2015-04-25 13:00:00	nan	NaN

```
df_missing.shape
```

```
(38, 6)
```

```
## Working with Date and Time
```

```
df_missing.shape
```

```
(38, 6)
```

```
## Working with Date and Time
```

```
def dateParser(s):
    """
    Function that takes a string in the format yyyy-mm-dd hh:mm:ss, and
    returns the same as a datetime object.
    """
    return datetime.datetime(int(s[0:4]), int(s[5:7]), int(s[8:10]), int(s[11:13]))
```

```
time.time()
```

time as a floating point number expressed in seconds since the epoch, in UTC.

The beginning of time in Python ...

```
t0 = time.time()
df['pickup_date_hour'] = df.pickup_datetime.apply(dateParser)
time.time() - t0
```

time as a floating point number expressed in seconds since the epoch, in UTC.

The beginning of time in Python ...

```
t0 = time.time()
df['pickup_date_hour'] = df.pickup_datetime.apply(dateParser)
time.time() - t0
```

```
64.49304580688477
```

```
beginning = df.pickup_date_hour.min()
end = df.pickup_date_hour.max()
print(beginning, end, end - beginning)
```

```
2014-09-01 00:00:00 2015-09-01 00:00:00 365 days 00:00:00
```

```
## Let us drop the pickup_datetime column
```

```
df = df.drop('pickup_datetime', axis=1)
```

```
df.head(10)
```

```
## Let us play with date and time in pandas
```

```
from pandas.tseries.holiday import USFederalHolidayCalendar

holidays = USFederalHolidayCalendar().holidays(beginning, end, return_name = True)
holidays
```

```
#Extract the weekday for each holiday
holidays.index.map(lambda x: x.strftime('%a'))
```

```
holidays
```

```
2014-09-01                      Labor Day
2014-10-13                   Columbus Day
2014-11-11                   Veterans Day
2014-11-27                   Thanksgiving
2014-12-25                      Christmas
2015-01-01                  New Years Day
2015-01-19    Dr. Martin Luther King Jr.
2015-02-16                 Presidents Day
2015-05-25                    MemorialDay
2015-07-03                      July 4th
dtype: object
```

```
#Extract the weekday for each holiday
holidays.index.map(lambda x: x.strftime('%a'))
```

```
Index(['Mon', 'Mon', 'Tue', 'Thu', 'Thu', 'Thu', 'Mon', 'Mon', 'Mon', 'Fri'], dtype='object')
```

```
t0 = time.time()
df['pickup_date'] = pd.Series(map(lambda x: x.astype('datetime64[D]'), df['pickup_date_hour'].values))
time.time() - t0
```

```
#Get month and year from pick up timestamp
df['year'] = df['pickup_date_hour'].dt.year
df['month'] = df['pickup_date_hour'].dt.month
```

```
#Get trip pick up day of the month
df['day'] = df['pickup_date_hour'].dt.day
```

```
df['hour'] = df['pickup_date_hour'].dt.hour
```

```
df['weekday'] = df['pickup_date_hour'].dt.dayofweek
```

```
df.weekday.value_counts()
```

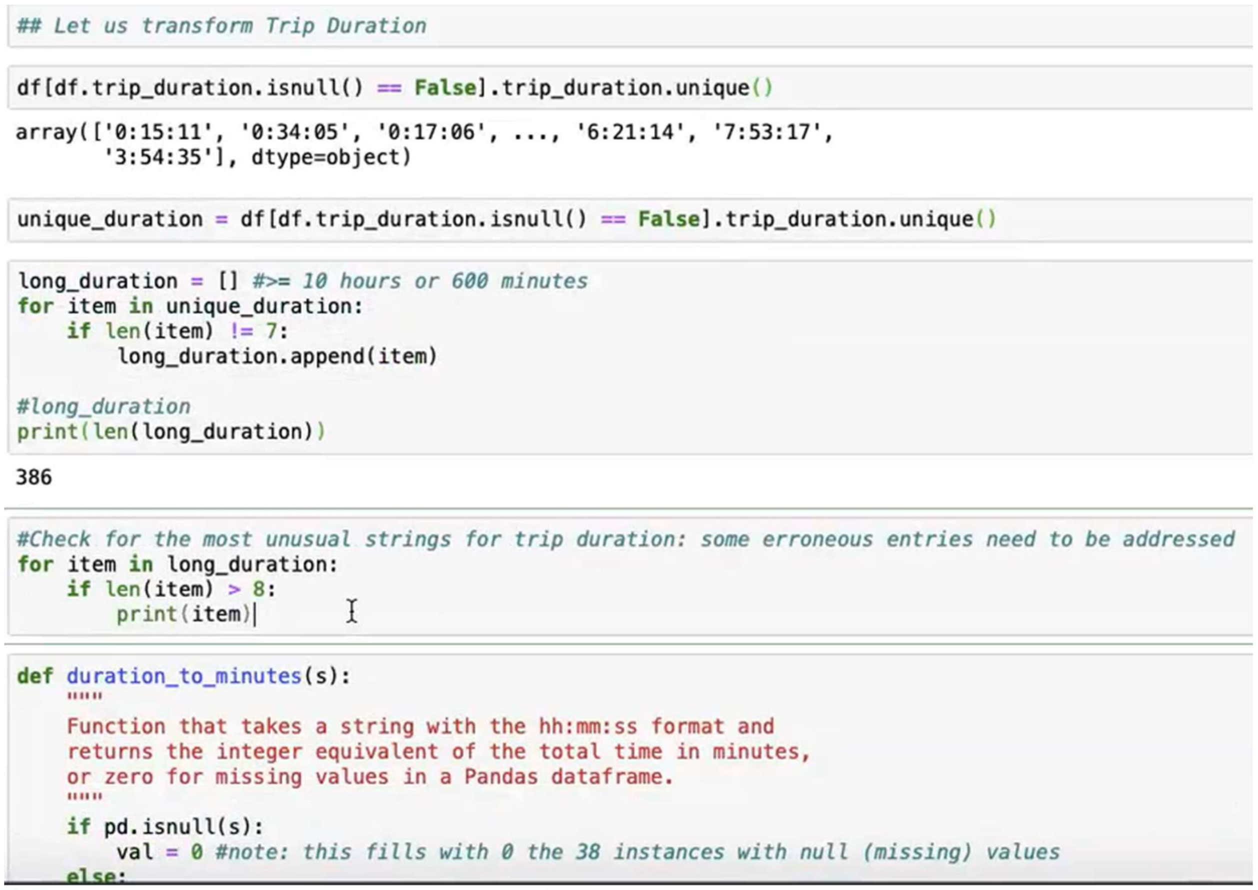
```
df.weekday.value_counts()
```

```
5    5142832
4    4805002
3    4625511
6    4331195
2    4301537
1    4010434
0    3709227
Name: weekday, dtype: int64
```

```
df.head()
```

	id	origin_taz	destination_taz	trip_distance	trip_duration	pickup_date_hour	pickup_date	year	month	day	hour	weekday
0	252581	7C	6A	4.2500	0:15:11	2014-09-01 09:00:00	2014-09-01	2014	9	1	9	0
1	252582	7B	15	10.1700	0:34:05	2014-09-01 18:00:00	2014-09-01	2014	9	1	18	0
2	252583	11	2A	4.0200	0:17:06	2014-09-01 17:00:00	2014-09-01	2014	9	1	17	0
3	252584	3B	4A	1.4600	0:06:32	2014-09-01 13:00:00	2014-09-01	2014	9	1	13	0
4	252585	2A	10	8.3100	0:26:17	2014-09-01 14:00:00	2014-09-01	2014	9	1	14	0

```
df.info()
```



```
## Let us transform Trip Duration
```

```
df[df.trip_duration.isnull() == False].trip_duration.unique()
```

```
array(['0:15:11', '0:34:05', '0:17:06', ..., '6:21:14', '7:53:17',
       '3:54:35'], dtype=object)
```

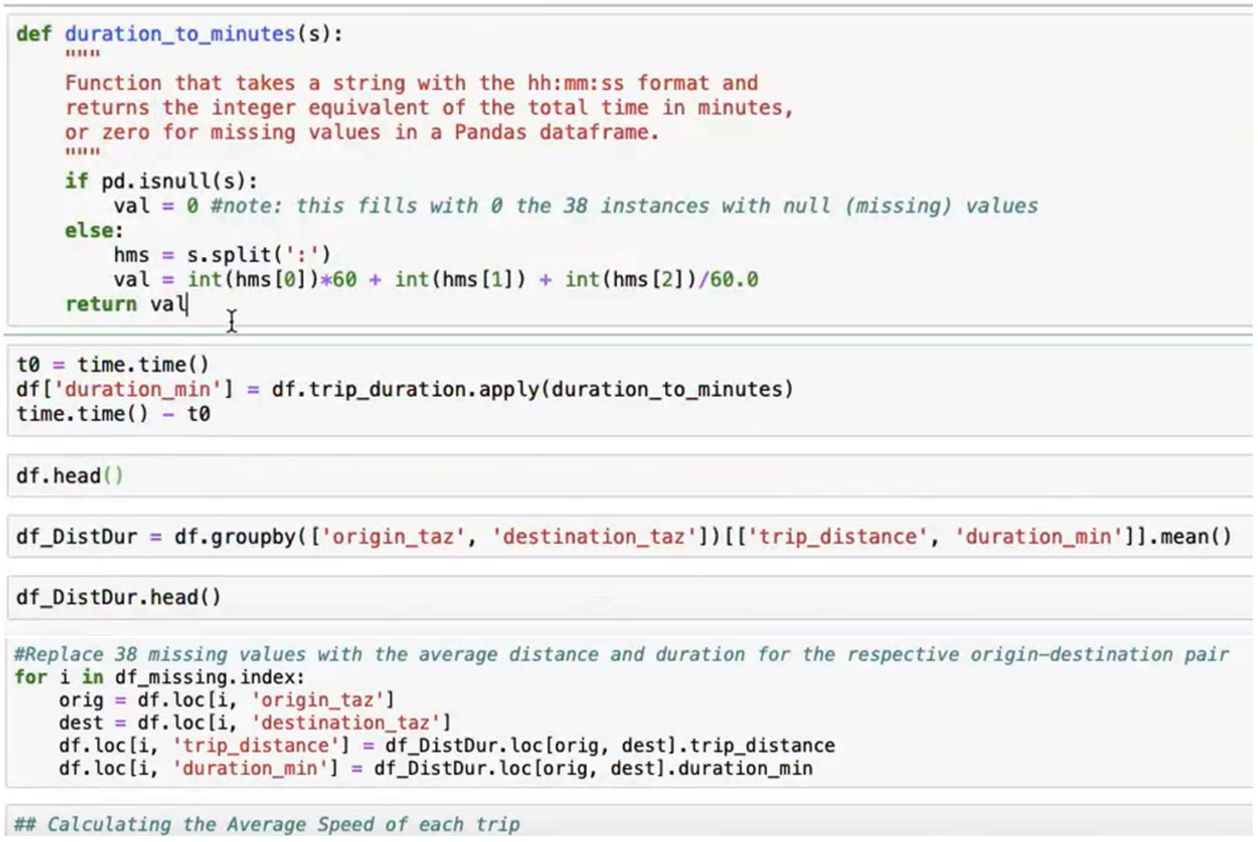
```
unique_duration = df[df.trip_duration.isnull() == False].trip_duration.unique()
```

```
long_duration = [] #>= 10 hours or 600 minutes
for item in unique_duration:
    if len(item) != 7:
        long_duration.append(item)

#long_duration
print(len(long_duration))
```

```
386
```

```
#Check for the most unusual strings for trip duration: some erroneous entries need to be addressed
for item in long_duration:
    if len(item) > 8:
        print(item)
```

```
def duration_to_minutes(s):
    """
    Function that takes a string with the hh:mm:ss format and
    returns the integer equivalent of the total time in minutes,
    or zero for missing values in a Pandas dataframe.
    """
    if pd.isnull(s):
        val = 0 #note: this fills with 0 the 38 instances with null (missing) values
    else:
```



```
def duration_to_minutes(s):
    """
    Function that takes a string with the hh:mm:ss format and
    returns the integer equivalent of the total time in minutes,
    or zero for missing values in a Pandas dataframe.
    """
    if pd.isnull(s):
        val = 0 #note: this fills with 0 the 38 instances with null (missing) values
    else:
        hms = s.split(':')
        val = int(hms[0])*60 + int(hms[1]) + int(hms[2])/60.0
    return val
```

```
t0 = time.time()
df['duration_min'] = df.trip_duration.apply(duration_to_minutes)
time.time() - t0
```

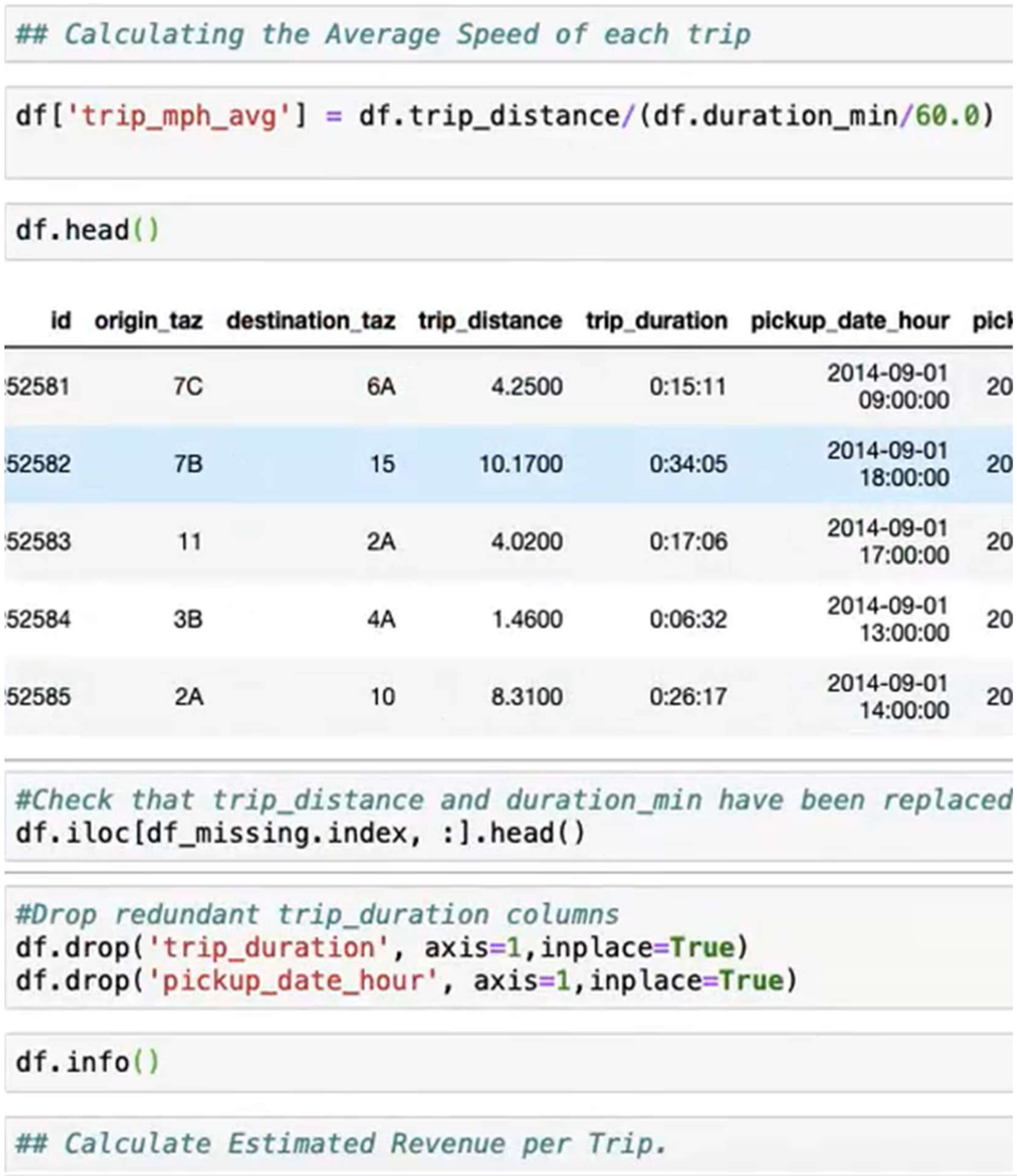
```
df.head()
```

```
df_DistDur = df.groupby(['origin_taz', 'destination_taz'])[['trip_distance', 'duration_min']].mean()
```

```
df_DistDur.head()
```

```
#Replace 38 missing values with the average distance and duration for the respective origin-destination pair
for i in df_missing.index:
    orig = df.loc[i, 'origin_taz']
    dest = df.loc[i, 'destination_taz']
    df.loc[i, 'trip_distance'] = df_DistDur.loc[orig, dest].trip_distance
    df.loc[i, 'duration_min'] = df_DistDur.loc[orig, dest].duration_min
```

```
## Calculating the Average Speed of each trip
```



```
## Calculating the Average Speed of each trip
```

```
df['trip_mph_avg'] = df.trip_distance/(df.duration_min/60.0)
```

```
df.head()
```

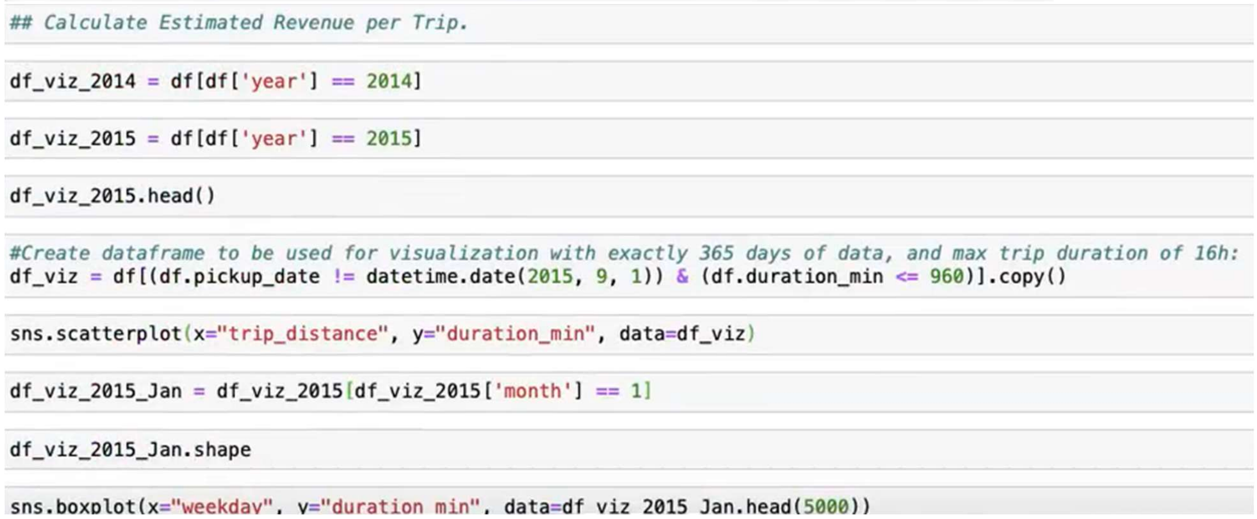
id	origin_taz	destination_taz	trip_distance	trip_duration	pickup_date_hour	pick
52581	7C	6A	4.2500	0:15:11	2014-09-01 09:00:00	20
52582	7B	15	10.1700	0:34:05	2014-09-01 18:00:00	20
52583	11	2A	4.0200	0:17:06	2014-09-01 17:00:00	20
52584	3B	4A	1.4600	0:06:32	2014-09-01 13:00:00	20
52585	2A	10	8.3100	0:26:17	2014-09-01 14:00:00	20

```
#Check that trip_distance and duration_min have been replaced
df.iloc[df_missing.index, :].head()
```

```
#Drop redundant trip_duration columns
df.drop('trip_duration', axis=1,inplace=True)
df.drop('pickup_date_hour', axis=1,inplace=True)
```

```
df.info()
```

```
## Calculate Estimated Revenue per Trip.
```



```
## Calculate Estimated Revenue per Trip.
```

```
df_viz_2014 = df[df['year'] == 2014]
```

```
df_viz_2015 = df[df['year'] == 2015]
```

```
df_viz_2015.head()
```

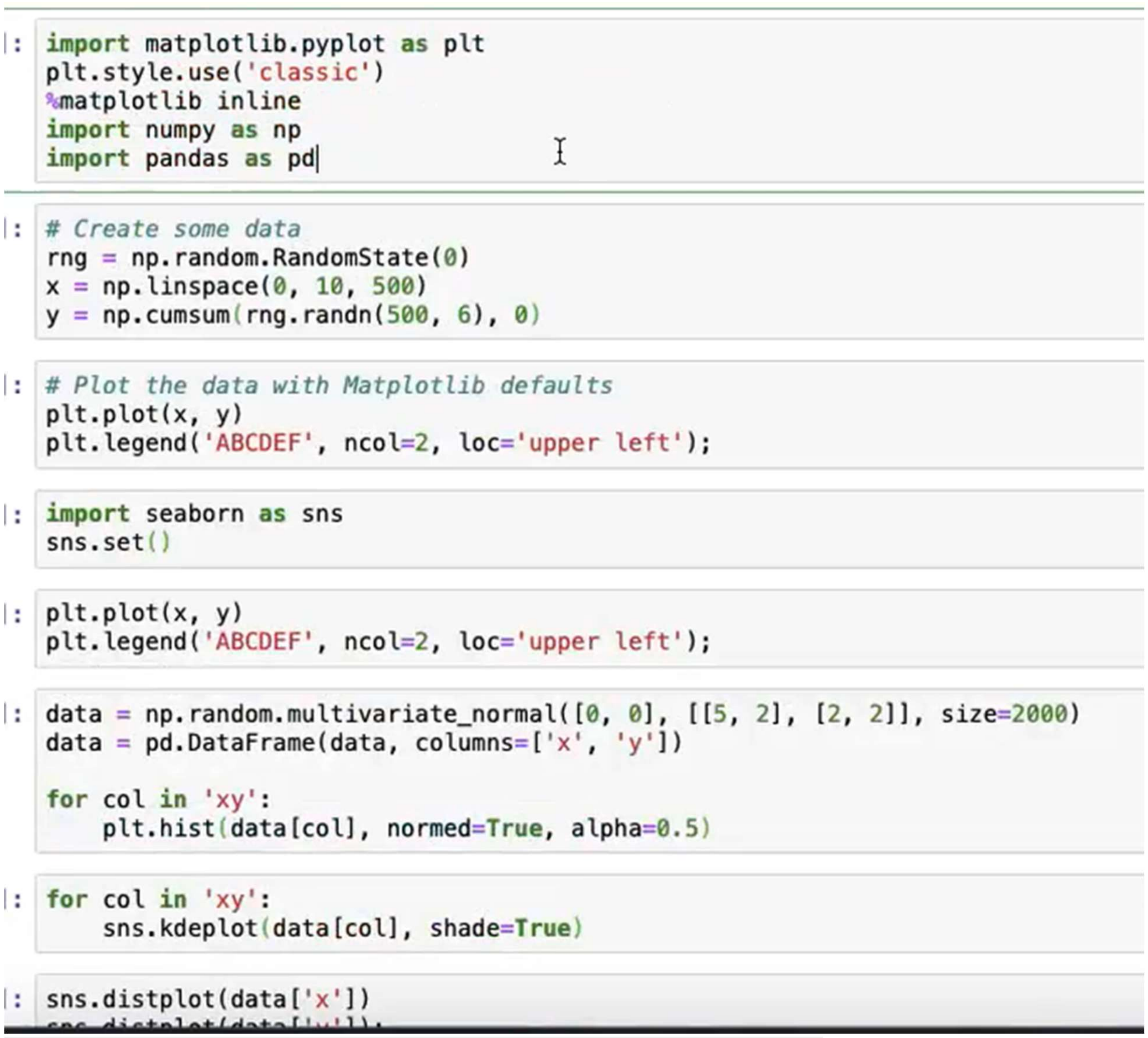
```
#Create dataframe to be used for visualization with exactly 365 days of data, and max trip duration of 16h:
df_viz = df[(df.pickup_date != datetime.date(2015, 9, 1)) & (df.duration_min <= 960)].copy()
```

```
sns.scatterplot(x="trip_distance", y="duration_min", data=df_viz)
```

```
df_viz_2015_Jan = df_viz_2015[df_viz_2015['month'] == 1]
```

```
df_viz_2015_Jan.shape
```

```
sns.boxplot(x="weekday", y="duration_min", data=df_viz_2015_Jan.head(5000))
```



```
import matplotlib.pyplot as plt
plt.style.use('classic')
%matplotlib inline
import numpy as np
import pandas as pd
```

```
# Create some data
rng = np.random.RandomState(0)
x = np.linspace(0, 10, 500)
y = np.cumsum(rng.randn(500, 6), 0)
```

```
# Plot the data with Matplotlib defaults
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left');
```

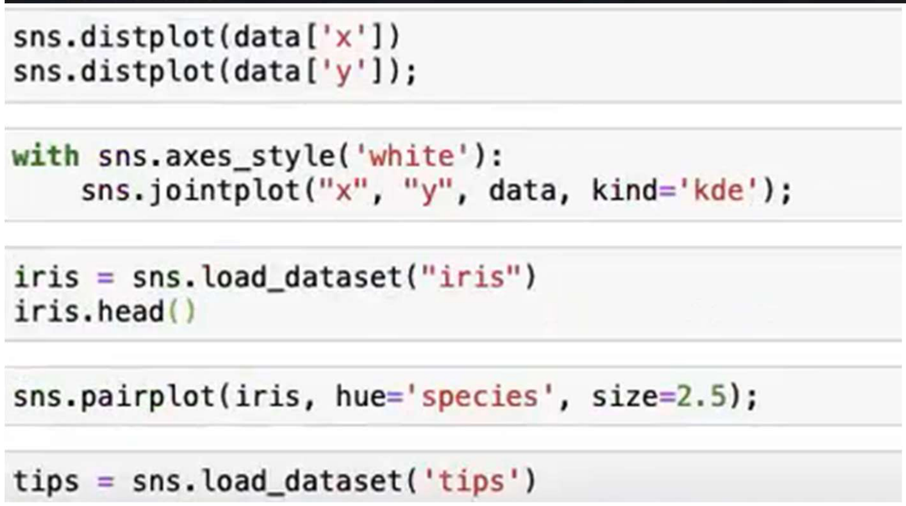
```
import seaborn as sns
sns.set()
```

```
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left');
```

```
data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])

for col in 'xy':
    plt.hist(data[col], normed=True, alpha=0.5)
```

```
for col in 'xy':
    sns.kdeplot(data[col], shade=True)
```

```
sns.distplot(data['x'])
```



```
sns.distplot(data['x'])
sns.distplot(data['y']);
```

```
with sns.axes_style('white'):
    sns.jointplot("x", "y", data, kind='kde');
```

```
iris = sns.load_dataset("iris")
iris.head()
```

```
sns.pairplot(iris, hue='species', size=2.5);
```

```
tips = sns.load_dataset('tips')
```

```python
sns.pairplot(iris, hue='species', size=2.5);
```

```
/Users/a137342/anaconda3/lib/python3.6/site-packages/seaborn/axisgrid.py:20
s been renamed to `height`; pleaes update your code.
  warnings.warn(msg, UserWarning)
```

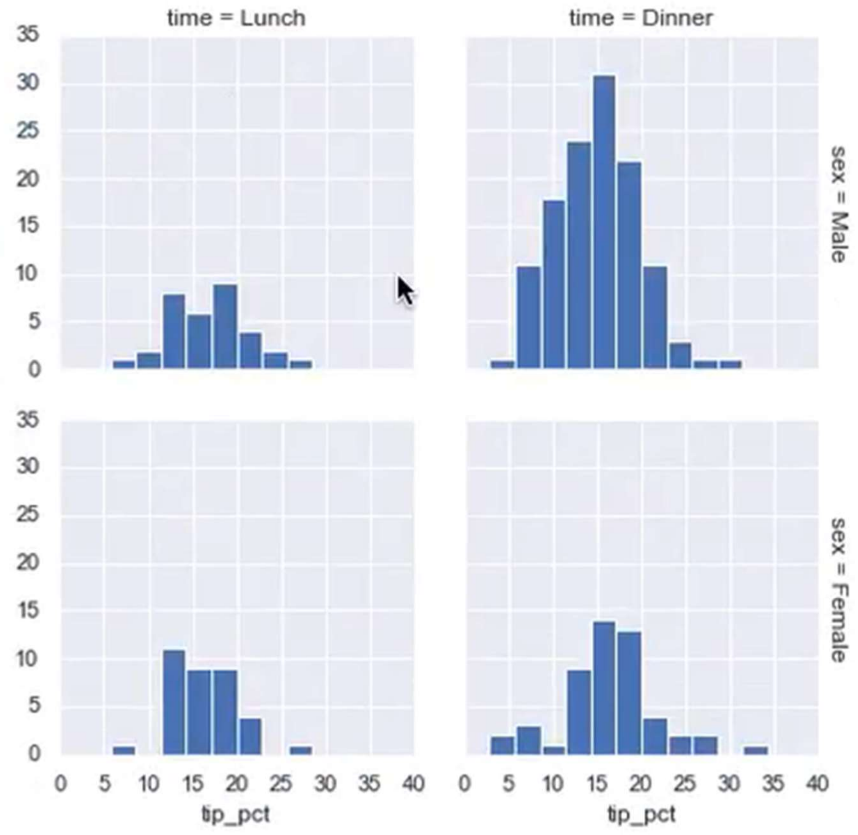
```python
tips = sns.load_dataset('tips')
tips.head()
```

```python
tips['tip_pct'] = 100 * tips['tip'] / tips['total_bill']

grid = sns.FacetGrid(tips, row="sex", col="time", margin_titles=True)
grid.map(plt.hist, "tip_pct", bins=np.linspace(0, 40, 15));
```

```python
with sns.axes_style(style='ticks'):
    g = sns.factorplot("day", "total_bill", "sex", data=tips, kind="box")
    g.set_axis_labels("Day", "Total Bill");
```

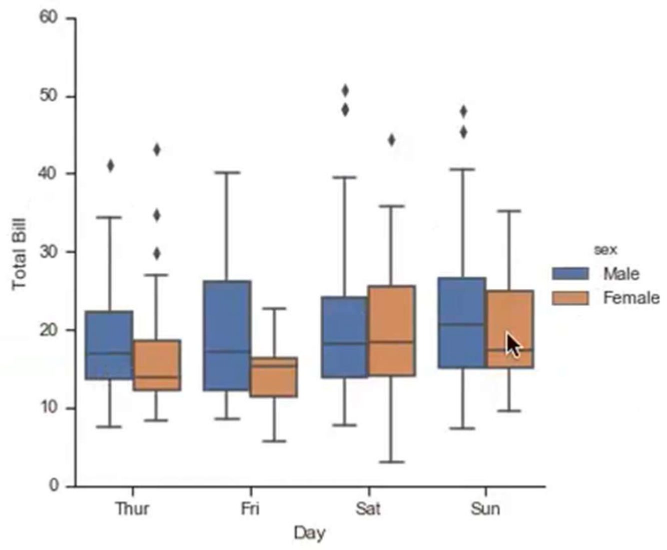
```python
sns.jointplot("total_bill", "tip", data=tips, kind='reg');
```

```python
tips['tip_pct'] = 100 * tips['tip'] / tips['total_bill']

grid = sns.FacetGrid(tips, row="sex", col="time", margin_titles=True)
grid.map(plt.hist, "tip_pct", bins=np.linspace(0, 40, 15));
```

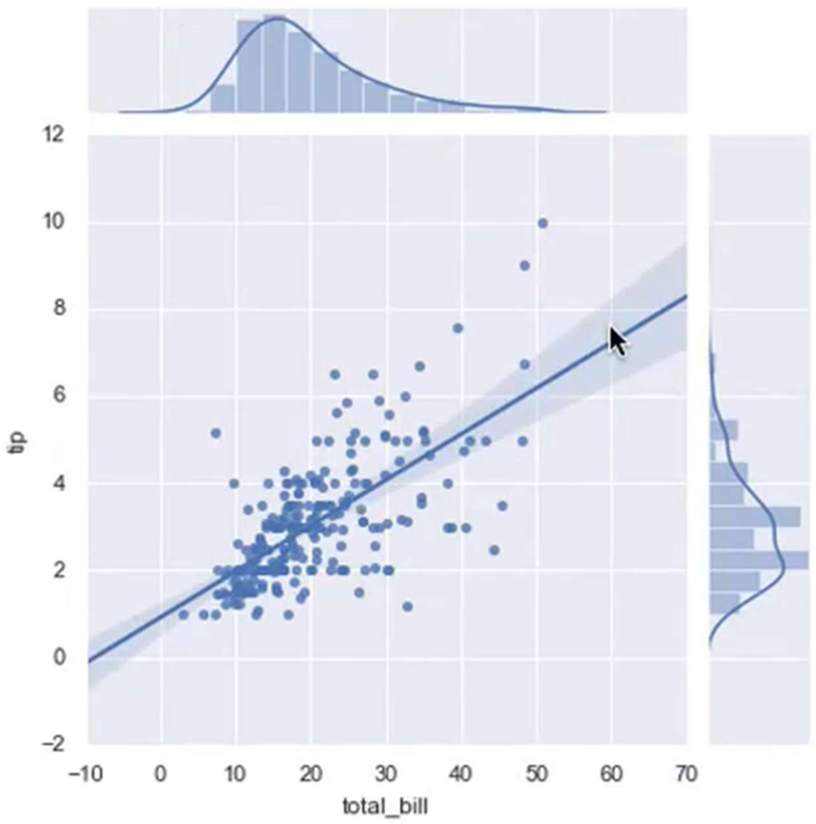


```
with sns.axes_style(style='ticks'):
    g = sns.factorplot("day", "total_bill", "sex", data=tips, kind="box")
    g.set_axis_labels("Day", "Total Bill");
```

```
/Users/a137342/anaconda3/lib/python3.6/site-packages/seaborn/categorical.py:3666: UserWarni
ction has been renamed to `catplot`. The original name will be removed in a future release.
. Note that the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplo
  warnings.warn(msg)
```

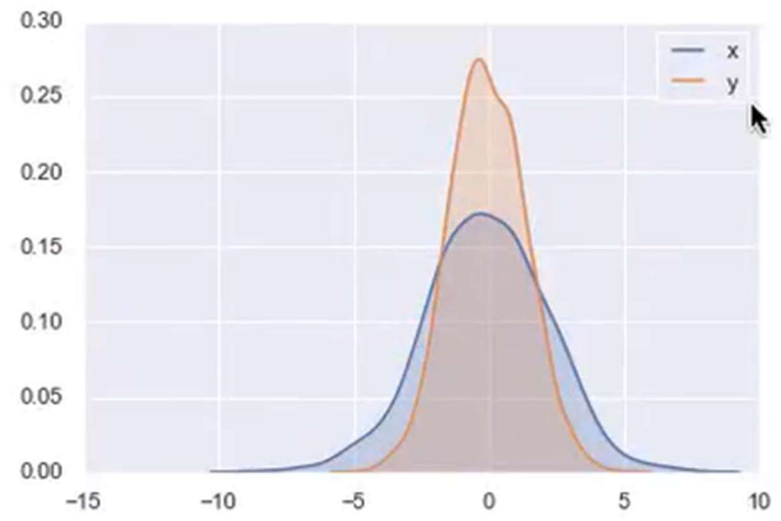


```
sns.jointplot("total_bill", "tip", data=tips, kind='reg');
```

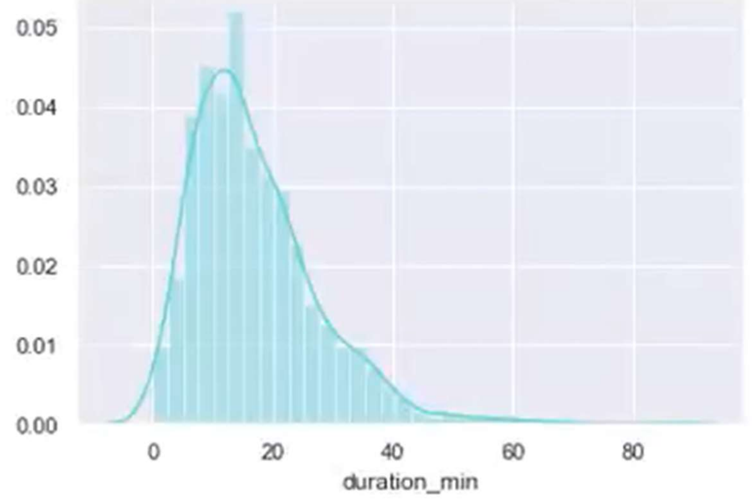


```
for col in 'xy':
    sns.kdeplot(data[col], shade=True)
```

```
/Users/a137342/anaconda3/lib/python3.6/site-packages/scipy/stat
quence for multidimensional indexing is deprecated; use `arr[tu
will be interpreted as an array index, `arr[np.array(seq)]`, wh
sult.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) /
```



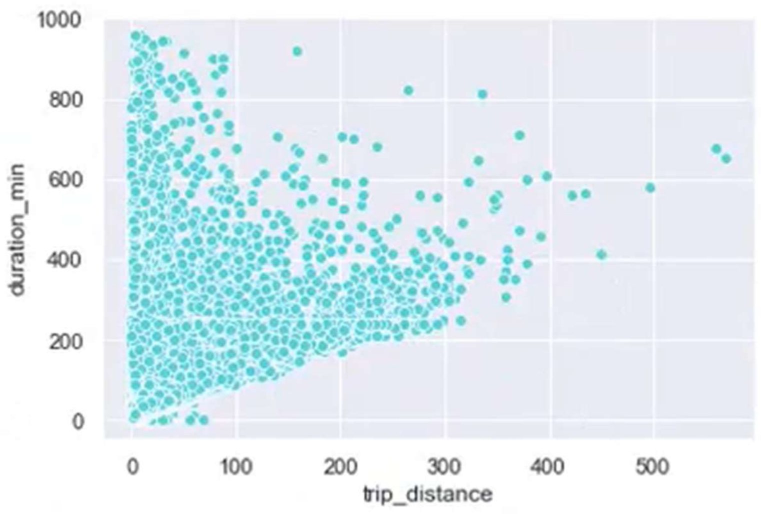
```
5]: sns.distplot(df_viz_2015_Jan['duration_min'].head(1000))
5]: <matplotlib.axes._subplots.AxesSubplot at 0x1a19e11080>
```



```
7]: sns.countplot(df_viz_2015_Jan.weekday)
```

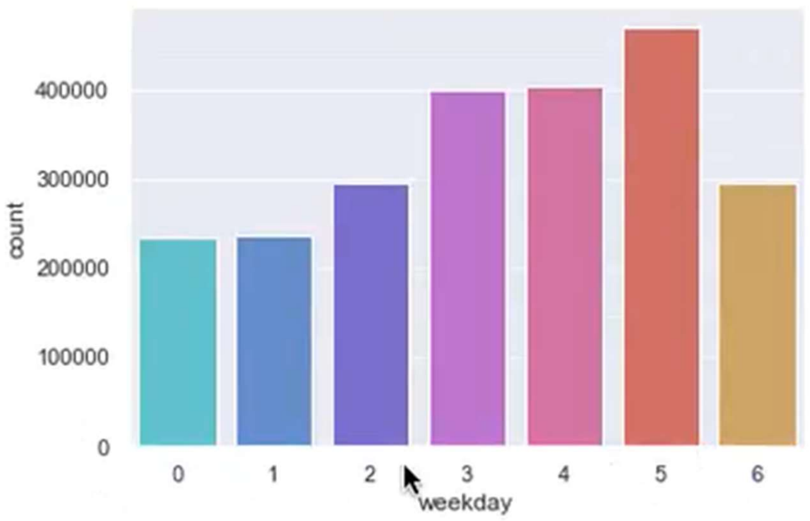
```
sns.scatterplot(x="trip_distance", y="duration_min", data=df_viz)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11605b978>
```



```
sns.countplot(df_viz_2015_Jan.weekday)
```

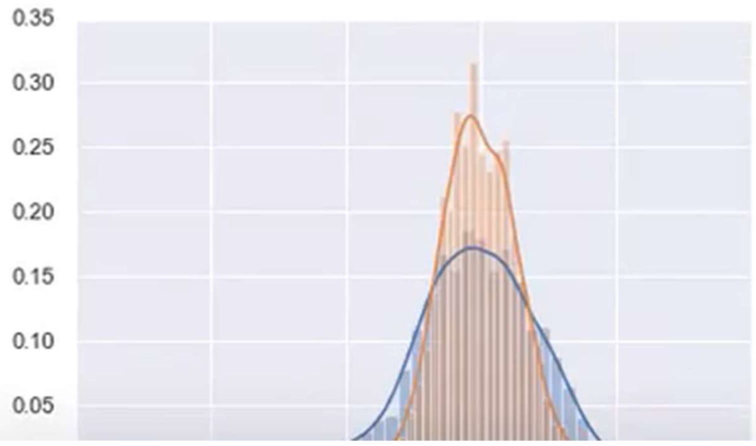
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a19f36c50>
```



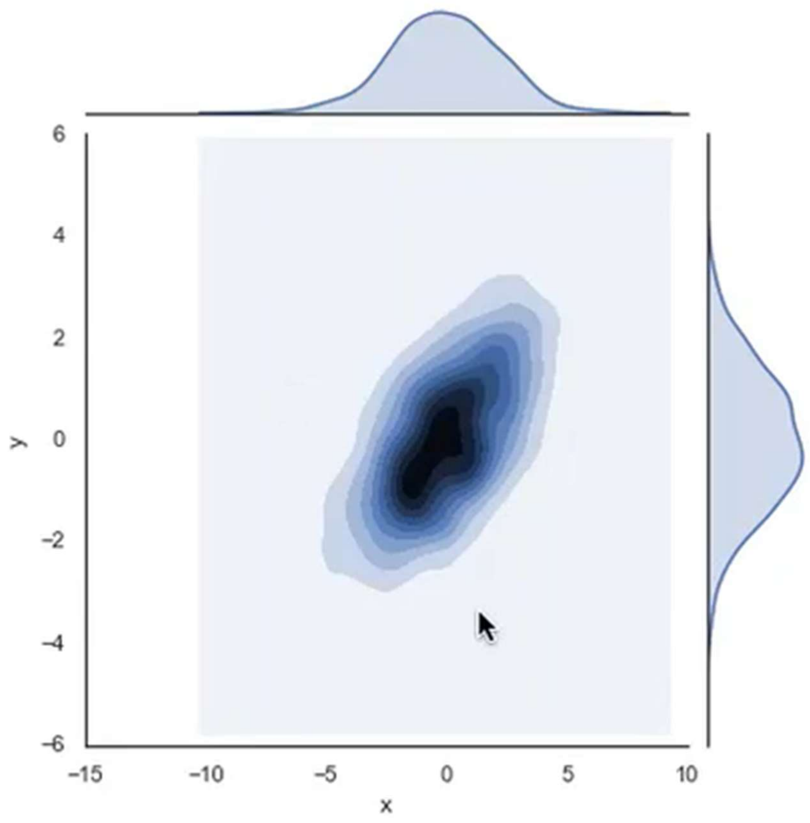
```
sns.countplot(df_viz_2015.month)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1172b1eb8>
```

```
sns.distplot(data['x'])
sns.distplot(data['y']);
```

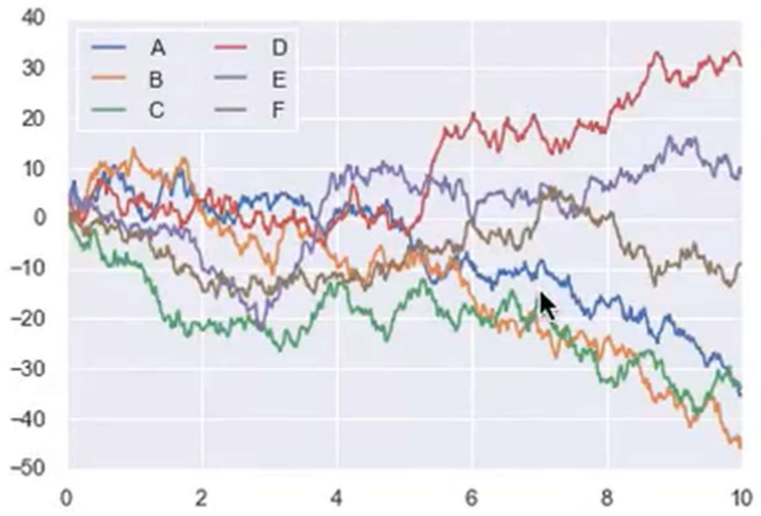


```
with sns.axes_style('white'):
    sns.jointplot("x", "y", data, kind='kde');
```



```
import seaborn as sns
sns.set()
```

```
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left');
```


A
B
C
D
E
F
40
30
20
10
0
−10
-20
-30
-40
-50
0
2
4
6
8
10