# Intro to Data Manipulation and Linear Regression in Python

## Setup

```
# install required packages
import subprocess

def check_installation(package_name):
    try:
        __import__(package_name)
        print(f"{package_name} is installed.")
    except ImportError:
        print(f"{package_name} is not installed. Installing now...")
        subprocess.check_call(["python", "-m", "pip", "install", package_name])

check_installation('numpy')
check_installation('pandas')
check_installation('statsmodels')
check_installation('linearmodels')
check_installation('stargazer')
```

## Instructions to follow along

It is recommended to follow along with the code examples during the lecture. All the codes presented in the slides are available in the `pythonmetrics.ipynb` file that you can download by:

- **Option 1**: Running `git pull` in the terminal from inside the repository folder we used for the class on web scraping.
- **Option 2**: Downloading the notebook from Blackboard in the repository folder.

## Intro

### Our Goals for Today:

1. Intro to numerical computing with NumPy
2. Basics of data manipulation with Pandas
3. OLS and IV Regression in Python

## Intro to numerical computing with NumPy

### What is NumPy?

- `NumPy` is the fundamental library for scientific computing in Python.
- `NumPy` adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- A **numpy array** is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers.
- Numpy arrays can be **one-dimensional** or **n-dimensional**.

### Create Numpy Arrays

- From a list: `np.array([1, 2, 3])`.
- Array of Zeros: `np.zeros(shape=(d0, d1, ...))`.
- Array of Ones: `np.ones(shape=(d0, d1, ...))`.
- Array of random numbers: `np.random.rand(d0, d1, ...)`.
- Array of evenly spaced integers: `np.arange(start,stop,step)`.
- Array of evenly spaced numbers: `np.linspace(start,stop,num)`.

```python
import numpy as np

array1 = np.zeros(5)
print('Array of Zeros:', array1)
array2 = np.random.rand(3, 4).round(2)
print('Array of Random Numbers:\n', array2)
```

```python
array3 = np.arange(5)
print('Array of Evenly Spaced Integers:', array3)
array4 = np.linspace(0, 1, 5)
print('Array of Evenly Spaced Numbers:', array4)
```

```
Array of Zeros: [0. 0. 0. 0. 0.]
Array of Random Numbers:
 [[0.75 0.15 0.73 0.43]
 [0.12 0.93 0.01 0.52]
 [0.71 0.37 0.31 0.1 ]]
Array of Evenly Spaced Integers: [0 1 2 3 4]
Array of Evenly Spaced Numbers: [0.   0.25 0.5  0.75 1.  ]
```

### Main Attributes of Numpy Arrays

- `ndim`: the number of dimensions of the array.
- `shape`: the size of the array along each dimension.
- `size`: the total number of elements of the array.
- `dtype`: the data type of the array (see list).

```python
array = np.random.rand(3, 4)
print('Number of Dimensions:', array.ndim)
print('Shape:', array.shape)
print('Size:', array.size)
print('Dtype:', array.dtype)
```

```
Number of Dimensions: 2
Shape: (3, 4)
Size: 12
Dtype: float64
```

### Main Methods of one-dimensional Numpy Arrays

- **Reshape** the array with `reshape()`.
- **Transpose** the array with `transpose()`.
- You can use a number of **aggregate functions** on numpy arrays (e.g. `sum()`, `mean()`, `std()`, `var()`, `max()`, `min()`, `median()`, `quantile()`, `argmax()`, `argmin()`).

```python
my_array = np.arange(4)
print('Reshaped Array:\n', my_array.reshape(2, 2))
print('Transposed Array:\n', my_array.transpose())
print('Sum of Array:', my_array.sum())
```

```
Reshaped Array:
 [[0 1]
 [2 3]]
Transposed Array:
 [0 1 2 3]
Sum of Array: 6
```

### Main Methods of n-dimensional Numpy Arrays

- You can use the methods of one-dimensional numpy arrays.
  - You can **specify the axis** along which to compute the aggregate function (0 for rows, 1 for columns).
- To **flatten** a n-dimensional numpy array, use the method `flatten()`.
- You can perform **linear algebra operations** with functions from `numpy.linalg` submodule. For instance:
  - Invert a two-dimensional array with the function `linalg.inv()`.
  - Compute the determinant of a two-dimensional array with the function `linalg.det()`.
  - Compute the eigenvalues and eigenvectors of a two-dimensional array with the function `linalg.eig()`.

```python
my_array = np.arange(1, 5).reshape(2, 2)

# sum along columns
print('Sum along columns:', my_array.sum(axis=0))

# flatten the array
print('Flattened Array:', my_array.flatten())

# invert the array
print('Inverted Array:\n', np.linalg.inv(my_array))
```

```python
# compute the determinant
print('Determinant of Array:', np.linalg.det(my_array).round(2))
```

```
Sum along columns: [4 6]
Flattened Array: [1 2 3 4]
Inverted Array:
 [[-2.   1. ]
 [ 1.5 -0.5]]
Determinant of Array: -2.0
```

### Basic Operations on Numpy Arrays

1. **Element-wise and scalar operations** on numpy arrays (e.g. `+`, `-`, `*`, `/`, `**`, `%`, `//`).
2. **Matrix operations** on numpy arrays (e.g. `np.matmul()`, `np.dot()`).
3. **Logical operations** on numpy arrays (e.g. `&`, `|`, `~`, `==`, `!=`, `>`, `<`, `>=`, `<=`).

```python
odimarray = np.arange(1, 5)
ndimarray = odimarray.reshape(2, 2)

print('Multiplication by a scalar:', odimarray * 2)
print('Sum element-wise:', odimarray + odimarray)
print('Dot Product:', np.dot(odimarray, odimarray))
print('Matrix multiplication:\n',
np.matmul(ndimarray, np.linalg.inv(ndimarray)).round(2))
print('Logical operations:\n', ndimarray > 2)
```

```
Multiplication by a scalar: [2 4 6 8]
Sum element-wise: [2 4 6 8]
Dot Product: 30
Matrix multiplication:
 [[1. 0.]
 [0. 1.]]
Logical operations:
 [[False False]
 [ True  True]]
```

### Slice Numpy Arrays

- One-dimensional numpy arrays can be **sliced** using the following syntax: `array[start:stop:step]`.
- Alternatively, use boolean arrays (i.e. **masks**): `array[array > 0]`.
- N-dimensional numpy arrays can be **sliced along every dimension**. Use "`:`" to select all elements along a dimension. Examples:
  - `array[start_1:stop_1:step_1,start_2:stop_2:step_2]`
  - `array[mask1, :]`.

```
# extract even numbers
print('Even Numbers:', np.arange(8)[::2])

# extract even numbers with a mask
print('Even Numbers:', np.arange(8)[np.arange(8) % 2 == 0])

# extract first row
my_array = np.arange(1, 5).reshape(2, 2)
print('First row:', my_array[0, :])

# extract first column
print('First colum:', my_array[:, 0])

# extract the diagonal
print('Diagonal:', np.diag(my_array))
```

```
Even Numbers: [0 2 4 6]
Even Numbers: [0 2 4 6]
First row: [1 2]
First colum: [1 3]
Diagonal: [1 4]
```

## Pandas for Data Cleaning and Manipulation

### What is Pandas?

- `Pandas` is a library that provides high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

- It is built on top of `NumPy` and it is one of the most popular libraries for data manipulation in Python.
- It is particularly useful for working with **tabular data** (e.g. data stored in a spreadsheet, database, or CSV file).

### Pandas DataFrames and Series

- The two main data structures in Pandas are `Series` and `DataFrames`.
- A **`Series`** is a one-dimensional array of indexed data.
  - It can contain any type of data (e.g. integers, floats, strings, booleans, etc.).
- A **`DataFrame`** is a two-dimensional array of indexed data.
  - It can be thought of as a sequence of aligned `Series` objects that share the same index.
  - The two dimensions of a `DataFrame` are commonly referred to as **rows** (Axis 0) and **columns** (Axis 1).

### Load data with Pandas

- `Pandas` provides a set of methods to **load data from different sources** (e.g. csv, xls, dta, json, html, etc.).
- The most common method is **`read_csv()`**, which loads data from a CSV file into a `DataFrame`.
- Other useful methods are: `read_excel()`, `read_stata()`, `read_json()`, `read_html()`.
- In the first part of this lecture, we use the `XBox Ebay Auctions Dataset`.
- The dataset contains 1,861 bids from 93 Ebay auctions on XBox Consoles.
- Columns:
  - `auctionid`: unique identifier of an auction
  - `bid`: bid placed by a bidder
  - `bidtime`: days since the start of the auction
  - `bidder`: eBay username of the bidder
  - `bidderrate`: user rating on eBay
  - `openbid`: the opening bid set by the seller
  - `price`: the closing price

```python
# load auction data from Google Drive
import pandas as pd
url = 'https://drive.google.com/file/d/18kuZupHEijS-rJxhtxeLX6rwg2rVi9rE/view?usp=sharing'
path = 'https://drive.google.com/uc?export=download&id='+url.split('/')[-2]
df = pd.read_csv(path)
```

### Explore Pandas Dataframes

- View **general info** about the dataset with the method `df.info()`.
- View the **first (last) rows** of the dataset with the method `df.head()` (`df.tail()`).
- Get the **number of rows and columns** with the attribute `df.shape`.
- Get **column (row) names** with the attribute `df.columns` (`df.index`).
- Compute **numerical summary statistics** with the method `df.describe()`.
- Check number of **missing values** with `df.isna().sum()` and drop them with `df.dropna()`.
- Check number of **duplicates** with `df.duplicated().sum()` and drop them with `df.drop_duplicates()`.

```python
# print general information about the dataset
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1861 entries, 0 to 1860
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   auctionid    1861 non-null   int64  
 1   bid          1861 non-null   float64
 2   bidtime      1861 non-null   float64
 3   bidder       1849 non-null   object 
 4   bidderrate   1850 non-null   float64
 5   openbid      1861 non-null   float64
 6   price        1861 non-null   float64
dtypes: float64(5), int64(1), object(1)
memory usage: 101.9+ KB
None
```

```python
# print first 5 rows
print(df.head(5))
```

```
    auctionid     bid   bidtime       bidder  bidderrate  openbid  price
0  8211480551   52.99  1.201505    hanna1104        94.0    49.99  311.6
1  8211480551   50.99  1.203843      wrufai1        90.0    49.99  311.6
2  8211480551  101.99  1.204433      wrufai1        90.0    49.99  311.6
3  8211480551   57.00  1.708437  newberryhwt        14.0    49.99  311.6
4  8211480551  144.48  3.089711    miloo2005         3.0    49.99  311.6
```

```python
# print the number of rows and columns
print("This is the shape of the DataFrame:", df.shape)

# print number of missing values
print('Number of missing values:')
print(df.isna().sum())

# print number of duplicated obs
print('Number of duplicates values:')
print(df.duplicated().sum())
```

```
This is the shape of the DataFrame: (1861, 7)
Number of missing values:
auctionid      0
bid            0
bidtime        0
bidder        12
bidderrate    11
openbid        0
price          0
dtype: int64
Number of duplicates values:
0
```

```python
# print summ stats
print(df.describe().round(2))

# print summ stats (latex)
# print(df.describe().style.to_latex(hrules=True,
# position_float='centering'))
```

```
          auctionid      bid  bidtime  bidderrate  openbid    price
count  1.861000e+03  1861.00  1861.00     1850.00  1861.00  1861.00
mean   8.213079e+09    86.03     5.16       30.51    22.22   149.09
std    9.404018e+05    59.87     2.36      135.44    28.47    71.56
min    8.211481e+09     0.01     0.00       -1.00     0.01    28.00
25%    8.212339e+09    42.00     3.79        0.00     0.99   105.01
50%    8.212848e+09    80.00     6.51        4.00     9.99   132.50
75%    8.213516e+09   115.00     6.93       19.00    49.00   167.50
max    8.214889e+09   405.00     7.00     2736.00   175.00   405.00
```

### Main columns operations

- **Change a column type** (e.g. string, int, float, data) with the method `df['column_name'].astype('type`
- **Rename** columns with the method `df.rename(columns={'old_name': 'new_name'})`.
- **Drop** columns with the method `df.drop(columns=['column_name_1', 'column_name_2'])`.
- Multiple ways to **create a new column**. Examples:
  - By simple assignment: `df['new_column_name'] = ...`.
  - By using the `assign()` method: `df.assign(new_column_name = ...)`.

```
# cast auctionid as string
df['auctionid'] = df['auctionid'].astype(str)

# print columns names
print(df.columns.tolist())

# new col for the diff between openbid and price
df['delta_price'] =  df['price'] - df['openbid']

# rename target column as species
df.rename(columns={'bidderrate': 'user_rate'}, inplace=True)

# print new columns names
print(df.columns.tolist())
```

```
['auctionid', 'bid', 'bidtime', 'bidder', 'bidderrate', 'openbid', 'price']
['auctionid', 'bid', 'bidtime', 'bidder', 'user_rate', 'openbid', 'price', 'delta_price']
```

- **Sort columns** with the method `df.sort_values(by=['column_name_1', 'column_name_2'])`.
- **Lambda functions** are useful to apply a function to a column:
  - `df['column_name'].apply(lambda x: x**2)`.

- df['column_name'].apply(lambda x: x**2 if x > 0 else x).

- **Conditional assignment** (using Numpy): df['new_column_name'] = np.where(df['column_name'] > 0, 1, 0).

```
# assign unknown to missing bidders with np.where
df['bidder'] = np.where(df['bidder'].isna(),
                        'unknown',
                        df['bidder'])

# create flag for winning bid
df['is_winning_bid'] = np.where(df['bid'] == df['price'], 1, 0)

# square bid
df['bid_squared'] = df['bid'].apply(lambda x: x**2)
```

```
           bid  bidtime  user_rate  openbid   price  delta_price  \
count  1861.00  1861.00    1850.00  1861.00 1861.00      1861.00
mean     86.03     5.16      30.51    22.22  149.09       126.87
std      59.87     2.36     135.44    28.47   71.56        78.85
min       0.01     0.00      -1.00     0.01   28.00         1.99
25%      42.00     3.79       0.00     0.99  105.01        72.50
50%      80.00     6.51       4.00     9.99  132.50       112.52
75%     115.00     6.93      19.00    49.00  167.50       141.51
max     405.00     7.00    2736.00   175.00  405.00       404.99

       is_winning_bid  bid_squared
count         1861.00      1861.00
mean             0.06     10984.85
std              0.24     16445.72
min              0.00         0.00
25%              0.00      1764.00
50%              0.00      6400.00
75%              0.00     13225.00
max              1.00    164025.00
```

### Subset Pandas DataFrames

- Methods df.loc[] and df.iloc[] select **subsets of rows and columns**.
  - df.loc[] selects rows and columns **by label** (i.e. column names and row names).
  - df.iloc[] selects rows and columns **by position** (i.e. column and row numbers).

- Alternatively, select a single column with `df['column_name']` and multiple columns with `df[['column_name_1', 'column_name_2']]`.
- Note that column is generally extracted as `Series` object, but can be transformed into a **numpy array** with `df['column_name'].values` or a **list** with `df['column_name'].tolist()`.

```
# select the first 2 rows and the first 2 columns
print(df.iloc[:2, :2])

# alternative
first_two_cols = df.columns[:2]
print(df[first_two_cols].head(2))

# select the rows 10-15 and the three columns
print(df.loc[10:13, ['auctionid', 'bidder', 'bid']])
```

```
    auctionid    bid
0  8211480551  52.99
1  8211480551  50.99
    auctionid    bid
0  8211480551  52.99
1  8211480551  50.99
     auctionid      bidder    bid
10  8211480551      pkfury  306.6
11  8211480551     wrufai1  311.6
12  8211763485  wolomaster   10.0
13  8211763485    deucekdp   11.0
```

- Select **rows that match one condition** with `df[df['column_name'] > 0]`.
- Select **rows matching multiple conditions** with `df[(df['column_name_1'] > 0) & (df['column_name_2'].isin(['val1', 'val2']))]`.
- Alternatively, use the method `df.query('column_name > 0')`.
- Extract a **random subsample** with `df.sample(n=10)`.
- Two good practices:
  - use the method `df.copy()` to make sure you create an ***independent* copy** of the dataframe. Example: `df[df['column_name'] > 0].copy()`.
  - use the method `df.reset_index()` to **reset the row names**.

```
# create a dataframe with winning bids
df_winningbids = df[df['is_winning_bid'] == 1].copy()

# alternatively
# df_winningbids = df.query("is_winning_bid == 1").copy()

# some auctions have multiple winning bids, keep only one
df_winningbids.drop_duplicates(subset = ['auctionid'], inplace=True)

# select columns of interest
df_winningbids = df_winningbids[['auctionid', 'bidder']]
# rename bidder column
df_winningbids.rename(columns={'bidder': 'winning_bidder'},
                                inplace=True)
# reset row names
df_winningbids.reset_index(drop=True, inplace=True)
```

### Groupby Operations

- The `groupby()` method allows to **group rows** of a `DataFrame` together and **call aggregate functions**.
- Use the `df.groupby('column_name')` method to create a `DataFrameGroupBy` object.
- Then, you can apply an aggregate function to the groupby object. Example: `df_grouped.mean()`.
- The `agg()` method allows to **apply multiple aggregate functions** at once to a grouped DataFrame.
  - You can pass a dictionary to the `agg()` method to specify the aggregate functions (values) to apply to each column (key).
- The `transform()` method allows to apply a function to each group and assign the result to a new column of the original dataset.

```
# group df by auctionid and user
grouped_df = df.groupby(['auctionid'])

# group by auction and user and compute avg and std
auctions_max = grouped_df.agg({'bidtime':'max',
                                'delta_price': 'max',
                                'bidder': 'nunique'})
```

```
# rename columns and rows
auctions_max.columns = ['max_bidtime', 'max_delta_price','n_bidders']

auctions_max.reset_index(inplace=True)

# add a col with number of bids per user
df['n_bidders'] = grouped_df.transform('size')

print(auctions_max.head(2))
```

```
    auctionid  max_bidtime  max_delta_price  n_bidders
0  8211480551     6.997338           261.61          9
1  8211763485     6.999780           129.31         12
```

### Merge DataFrames

- The `merge()` method allows to merge two `DataFrame` objects together.
- There are four types of joins:
  - **Inner join**: only keep rows that match from both `DataFrame` objects.
  - **Left join**: keep all rows from the left `DataFrame` object and only keep rows that match from the right `DataFrame` object.
  - **Right join**: keep all rows from the right `DataFrame` object and only keep rows that match from the left `DataFrame` object.
  - **Outer join**: keep all rows from both `DataFrame` objects.
- The `merge()` method has the following arguments:
  - `left`: the left `DataFrame` object.
  - `right`: the right `DataFrame` object.
  - `how`: the type of join.
  - `on`: the column(s) to join on.

```
# add data on winning user
df_merged = pd.merge(left =df,
              right=df_winningbids,
              how='inner',
              on='auctionid')

# check we have the same number of rows
print(df.shape[0] == df_merged.shape[0])
```

```
True
```

### Concatenate DataFrames

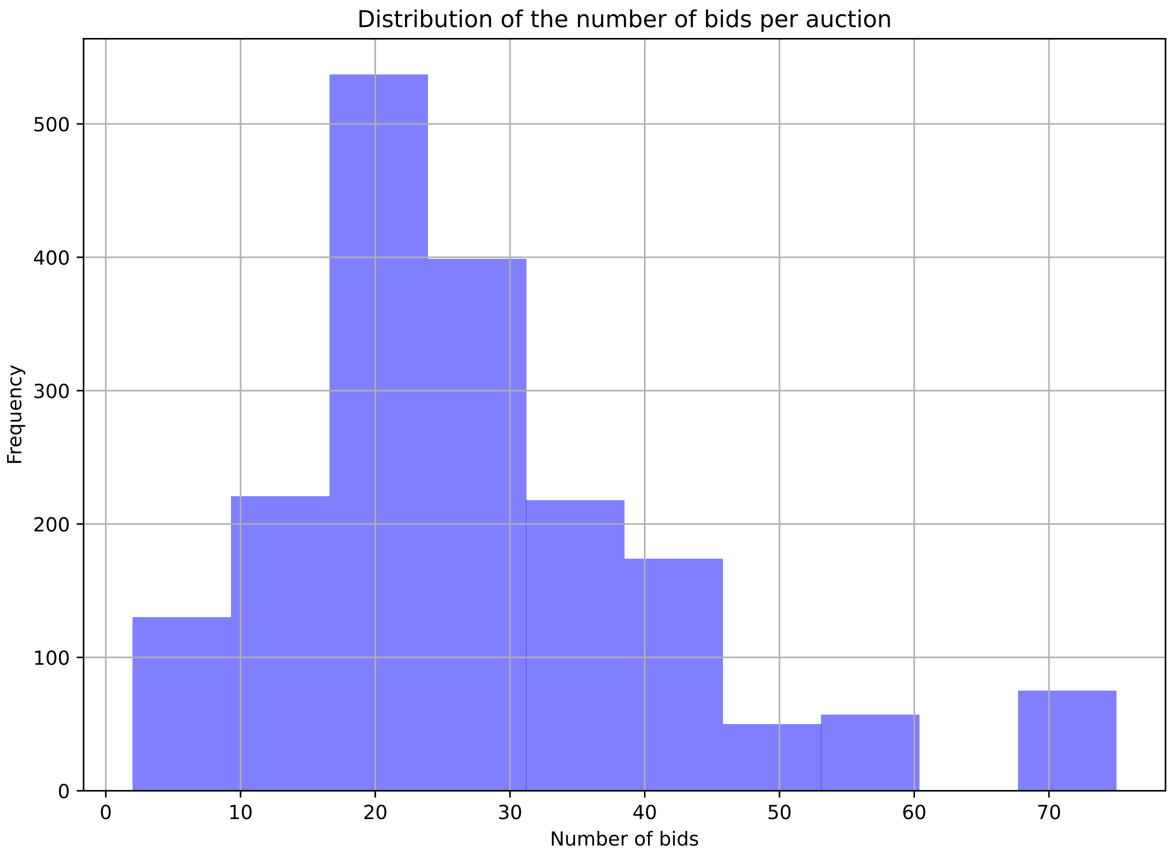
- The `concat()` method allows **to concatenate** two or more `DataFrame` objects together.
- The `axis` argument specifies the axis along which to concatenate the `DataFrame` objects (0 for rows, 1 for columns).

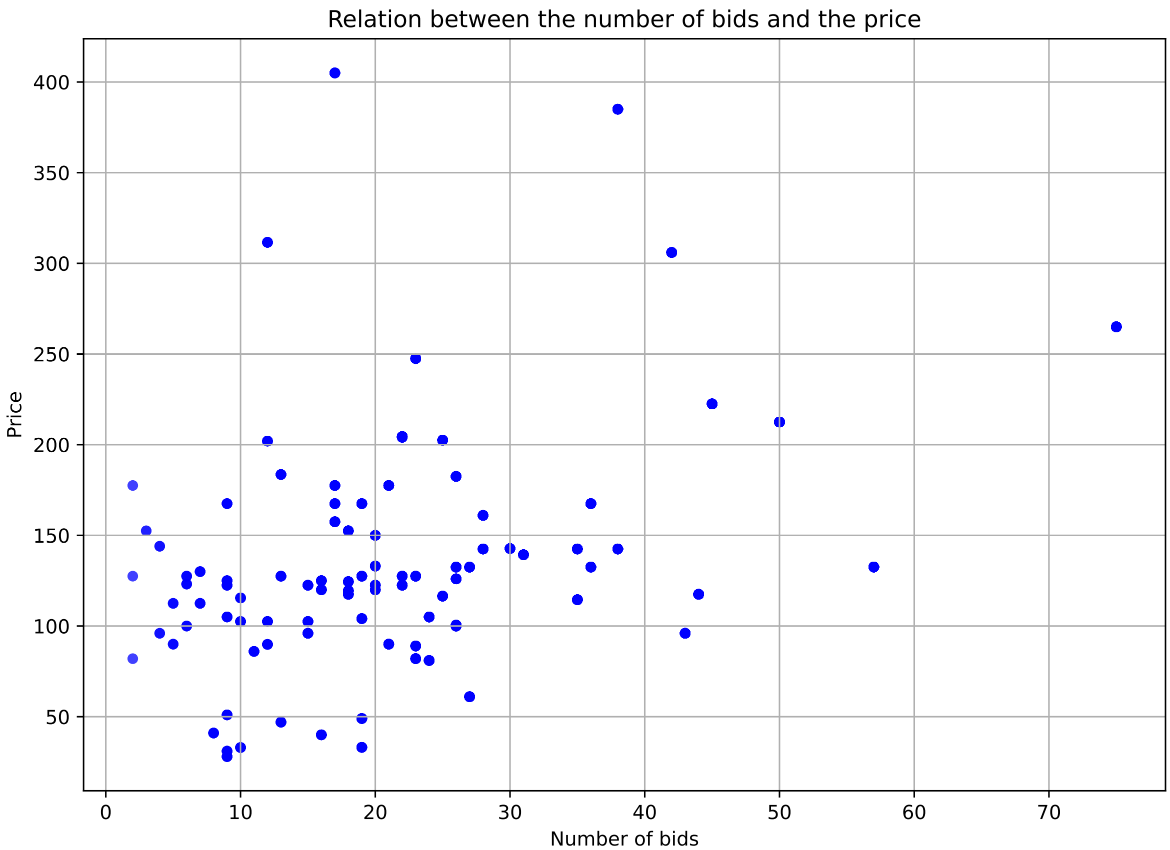
```
# create a list of dataframes
df_list = [df[df['is_winning_bid'] == i].copy()
                            for i in range(2)]

# concatenate row-wise the dataframes in the list
df_concat = pd.concat(objs=df_list, axis=0)

# check we have the same number of rows
print(df.shape[0] == df_concat.shape[0])

# delete the list of dataframes and df_concat
del df_list, df_concat
```

```
True
```

### Simple Plotting with Pandas

- The `plot()` method allows to plot a `DataFrame` object.
- The `kind` argument specifies the type of plot (e.g. `'line'`, `'bar'`, `'hist'`, `'box'`, `'scatter'`, `'pie'`, `'hexbin'`).
- The `x` and `y` arguments specify the columns to plot.
- Various arguments allow to customize the plot (e.g. `title`, `xlabel`, `ylabel`, `color`, `alpha`, `grid`, `legend`, `rot`).
- Very useful to quickly explore the data.


Distribution of the number of bids per auction
Frequency
Number of bids
0
100
200
300
400
500
0
10
20
30
40
50
60
70




## OLS and IV Regression in Python

### Main Libraries for Linear Regression in Python

- `statsmodels` is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration.
- `linearmodels` is a Python module that provides classes and functions for the estimation of linear models using OLS and IV, tests and statistics for linear models, and panel data models (and much more)
- Other libraries (extremely popular in ML) like `scikit-learn`

### Running Linear Regressions with Statsmodels

- The `statsmodels` module provides the `ols()` method to run linear regressions.
- You can define the specification by using the `formula` argument. E.g.: `Y ~ X1 + X2`
- To run the model, use the `fit()` method.
- Results can be explored with the `summary()` method.
- Get more details on the available methods and attributes here.
    - You can also explore available methods and attributes with `dir(your_model)`.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        max_delta_price   R-squared:                       0.355
Model:                            OLS   Adj. R-squared:                  0.348
Method:                 Least Squares   F-statistic:                     50.01
Date:                Tue, 31 Oct 2023   Prob (F-statistic):           3.04e-10
Time:                        16:06:14   Log-Likelihood:                -512.15
No. Observations:                  93   AIC:                             1028.
Df Residuals:                      91   BIC:                             1033.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.1603     15.269     -0.011      0.992     -30.490      30.169
n_bidders     11.4102      1.613      7.072      0.000       8.205      14.615
==============================================================================
Omnibus:                       64.397   Durbin-Watson:                   2.075
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              311.822
Skew:                           2.271   Prob(JB):                     1.94e-68
Kurtosis:                      10.736   Cond. No.                         23.3
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

- You can access:
    - the **coefficients** of a linear regression with the `params` attribute.
    - the **fitted values** of a linear regression with the `fittedvalues` attribute.
    - the **residuals** of a linear regression with the `resid` attribute.

```python
# get the coefficients
print("Coefficients:\n", model.params)

# store residuals and fitted values in the DataFrame
df['fitted_values'] = model.fittedvalues
df['residuals'] = model.resid
```

```python
print(df[['auctionid', 'bidder', 'residuals', 'fitted_values']].head(2))
```

```
Coefficients:
 Intercept    -0.160349
n_bidders    11.410153
dtype: float64
    auctionid     bidder   residuals  fitted_values
0  8211480551  hanna1104  159.078976     102.531024
1  8211480551    wrufai1   -7.451481     136.761481
```

### Advanced Options for Linear Regressions with Statsmodels

- To use robust or clustered covariance, use the method `get_robustcov_results()`.
  - Choose the covariance type with the `cov_type` argument (see types)
- You can add fixed effects within the formula with the `C()` function. See more details `here`.
  - In presence of high-dimensional fixed effects, it is better to use other libraries (e.g. `pyHDFE`). However, Stata (reghdfe) and R (fixest) implementations are generally faster.
- You can add interactions terms within the formula with the `*` operator.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    bid   R-squared:                       0.760
Model:                            OLS   Adj. R-squared:                  0.750
Method:                 Least Squares   F-statistic:                     47.84
Date:                Tue, 31 Oct 2023   Prob (F-statistic):           3.71e-26
Time:                        16:06:14   Log-Likelihood:                -564.52
No. Observations:                 117   AIC:                             1141.
Df Residuals:                     111   BIC:                             1158.
Df Model:                           5
Covariance Type:                  HC1
==============================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
Intercept                     97.8826     18.080      5.414      0.000      62.056     133.709
C(auctionid)[T.8211763485]  -119.5862     16.900     -7.076      0.000    -153.074     -86.098
C(auctionid)[T.8211851222]  -131.3167     18.754     -7.002      0.000    -168.479     -94.154
C(auctionid)[T.8212110388]  -161.1533     17.833     -9.037      0.000    -196.491    -125.816
C(auctionid)[T.8212116757]   -98.5913     18.624     -5.294      0.000    -135.496     -61.687
bidtime                       19.0869      1.948      9.796      0.000      15.226      22.948
==============================================================================
Omnibus:                        6.269   Durbin-Watson:                   0.679
Prob(Omnibus):                  0.044   Jarque-Bera (JB):                5.763
Skew:                           0.473   Prob(JB):                       0.0560
Kurtosis:                       3.536   Cond. No.                         44.0
```

```
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
```

## Diagnostic Tests

- Statsmodels provides a number of diagnostic tests for linear regressions.
- You can run diagnostic tests for linear regressions with the **stats module**. Examples:
  - Run the **Breusch-Pagan test** for heteroskedasticity with the `diagnostic.het_breuschpagan()` function.
  - Run the **White test** for heteroskedasticity with the `diagnostic.het_white()` function.
  - Run the **Jarque-Bera test** for normality with the `stattools.jarque_bera()` function.
  - See more details here.

```python
# import modules for tests
import statsmodels.stats.diagnostic as diagnostic
import statsmodels.stats.stattools as stattools

# run the Breusch-Pagan test for heteroskedasticity
print("Results Breusch-Pagan test:",
diagnostic.het_breuschpagan(model2.resid, model2.model.exog))

# run the White test for heteroskedasticity
print("Results White test:",
diagnostic.het_white(model2.resid, model2.model.exog))

# run the Jarque-Bera test for normality
print("Results Jarque-Bera test:",
stattools.jarque_bera(model2.resid))
```

## IV Regressions with linearmodels

- The `linearmodels` module provides the `IV2SLS()` class to run IV regressions.
- You can define a formula to run an IV regression with the `from_formula()` method. The instrumented variable and the instruments should be included inside square brackets.

  - E.g.: `Y ~ 1 + [X1 ~ Z1 + Z2] + X2`

- You can run an IV regression with the `fit()` method.
- The estimated coefficients are stored in the `params` attribute.
- You can access the first stage results with the `first_stage` attribute.
- Many other useful methods and attributes, see details `here`.
  - For instance, the `sargan` attribute returns the results of the Sargan test of overidentification.
- In this section, we will use the `Angrist and Krueger (1991)` dataset.
- Vary famous paper in which the authors use the quarter of birth as an instrument for years of schooling and estimate the returns to education.
- The US compulsory attendance law dictates that children must be 6 years old by January 1 when they start school, leading to older entry for those born at the beginning of the year. Moreover, the law compels students to stay in school until age 16, resulting in individuals born later in the year having, on average, more education than those born earlier.
- See this tutorial for a more detailed explanation.

```
        log_wage  years_of_schooling  year_of_birth  quarter_of_birth  \
count  329509.00           329509.00       329509.0          329509.00
mean        5.90               12.77           34.6               2.51
std         0.68                3.28            2.9               1.11
min        -2.34                0.00           30.0               1.00
25%         5.64               12.00           32.0               2.00
50%         5.95               12.00           35.0               3.00
75%         6.26               15.00           37.0               3.00
max        10.53               20.00           39.0               4.00

       state_of_birth         q1         q2         q3         q4
count       329509.00  329509.00  329509.00  329509.00  329509.00
mean            30.69       0.25       0.24       0.26       0.25
std             14.22       0.43       0.43       0.44       0.43
min              1.00       0.00       0.00       0.00       0.00
25%             19.00       0.00       0.00       0.00       0.00
50%             34.00       0.00       0.00       0.00       0.00
75%             42.00       0.00       0.00       1.00       0.00
max             56.00       1.00       1.00       1.00       1.00
```

```python
# import IV2SLS
from linearmodels.iv import IV2SLS

# subset the dataset
ak91_red = ak91[(ak91['state_of_birth'].isin(range(6))) &
                (ak91['year_of_birth'].isin(range(30, 34)))].copy()
ak91_red.reset_index(drop=True, inplace=True)

# define the formula
full_formula = r'''
log_wage ~  1 + C(year_of_birth) + C(state_of_birth) +
[years_of_schooling ~ q1 + q2 + q3]'''

# run the model
modelIV = IV2SLS.from_formula(formula=full_formula, data=ak91_red)
modelIV = modelIV.fit(cov_type="unadjusted")

# print(modelIV.summary)
# print(modelIV.first_stage)
```

```
       First Stage Estimation Results
=============================================
                           years_of_schooling
---------------------------------------------
R-squared                              0.0180
Partial R-squared                      0.0017
Shea's R-squared                       0.0017
Partial F-statistic                    3.4019
P-value (Partial F-stat)               0.0170
Partial F-stat Distn                 F(3,5840)
==========================        ===========
Intercept                              11.478
                                     (87.494)
C(year_of_birth)[T.31.0]               0.1048
                                     (0.7770)
C(year_of_birth)[T.32.0]               0.2530
                                     (1.9186)
C(year_of_birth)[T.33.0]               0.6027
                                     (4.5502)
C(state_of_birth)[T.2.0]               2.0805
                                     (2.9816)
C(state_of_birth)[T.4.0]               1.5435
                                     (7.8886)
C(state_of_birth)[T.5.0]              -0.0752
                                    (-0.7531)
q1                                    -0.3556
                                    (-2.6821)
q2                                    -0.3266
                                    (-2.4281)
q3                                    -0.0972
                                    (-0.7334)
```

```
--------------------------------------------

T-stats reported in parentheses
T-stats use same covariance type as original model


                          IV-2SLS Estimation Summary
==============================================================================
Dep. Variable:               log_wage   R-squared:                     -0.0110
Estimator:                    IV-2SLS   Adj. R-squared:                -0.0123
No. Observations:                5850   F-statistic:                    62.906
Date:                Tue, Oct 31 2023   P-value (F-stat)                0.0000
Time:                        16:06:15   Distribution:                  chi2(7)
Cov. Estimator:            unadjusted

                                    Parameter Estimates
=========================================================================================
                          Parameter  Std. Err.     T-stat    P-value    Lower CI    Upper CI
-----------------------------------------------------------------------------------------
Intercept                    4.1112     0.7227     5.6890     0.0000      2.6948      5.5276
C(year_of_birth)[T.31.0]     0.0134     0.0284     0.4734     0.6359     -0.0422      0.0691
C(year_of_birth)[T.32.0]    -0.0144     0.0316    -0.4556     0.6487     -0.0764      0.0476
C(year_of_birth)[T.33.0]    -0.0634     0.0477    -1.3291     0.1838     -0.1568      0.0301
C(state_of_birth)[T.2.0]     0.1088     0.1943     0.5599     0.5756     -0.2721      0.4897
C(state_of_birth)[T.4.0]     0.0525     0.1059     0.4957     0.6201     -0.1550      0.2600
C(state_of_birth)[T.5.0]     0.0893     0.0210     4.2580     0.0000      0.0482      0.1304
years_of_schooling           0.1403     0.0640     2.1906     0.0285      0.0148      0.2657
=========================================================================================

Endogenous: years_of_schooling
Instruments: q1, q2, q3
Unadjusted Covariance (Homoskedastic)
Debiased: False
```

```python
# extract second stage coefficient
schooling_coef = modelIV.params['years_of_schooling']
print('Second Stage Schooling Coefficient:',
      np.round(schooling_coef, 2))

# store residuals and fitted values in the DataFrame
ak91_red['residuals'] = modelIV.resids
ak91_red['fitted_values'] = modelIV.fitted_values
```

```
Second Stage Schooling Coefficient: 0.14
```

## Output Results of Linear Regressions

- You can output the results of a linear regression in Latex with the `as_latex()` method.
- In alternative, you can use the Python version of the R stargazer package.

  - Very useful for outputting multiple regressions in Latex.

- Note: to compare the models created with IV2SLS, you can use the `compare()` method.

```
\begin{table}[!htbp] \centering
\begin{tabular}{@{\extracolsep{5pt}}lcc}
\\[-1.8ex]\hline
\hline \\[-1.8ex]
\\[-1.8ex] & (1) & (2) \\
\hline \\[-1.8ex]
 C(auctionid)[T.8211763485] & & -119.586$^{***}$ \\
& & (10.552) \\
 C(auctionid)[T.8211851222] & & -131.317$^{***}$ \\
& & (11.486) \\
 C(auctionid)[T.8212110388] & & -161.153$^{***}$ \\
& & (10.891) \\
 C(auctionid)[T.8212116757] & & -98.591$^{***}$ \\
& & (12.080) \\
 Intercept & -0.160$^{}$ & 97.883$^{***}$ \\
& (15.269) & (11.240) \\
 bidtime & & 19.087$^{***}$ \\
& & (1.630) \\
 n_bidders & 11.410$^{***}$ & \\
& (1.613) & \\
\hline \\[-1.8ex]
 Observations & 93 & 117 \\
 $R^2$ & 0.355 & 0.760 \\
 Adjusted $R^2$ & 0.348 & 0.750 \\
 Residual Std. Error & 60.273 (df=91) & 30.949 (df=111) \\
 F Statistic & 50.014$^{***}$ (df=1; 91) & 70.466$^{***}$ (df=5; 111) \\
\hline
\hline \\[-1.8ex]
\textit{Note:} & \multicolumn{2}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05; $^{***}$p$<$0.01} \\
\end{tabular}
\end{table}
```