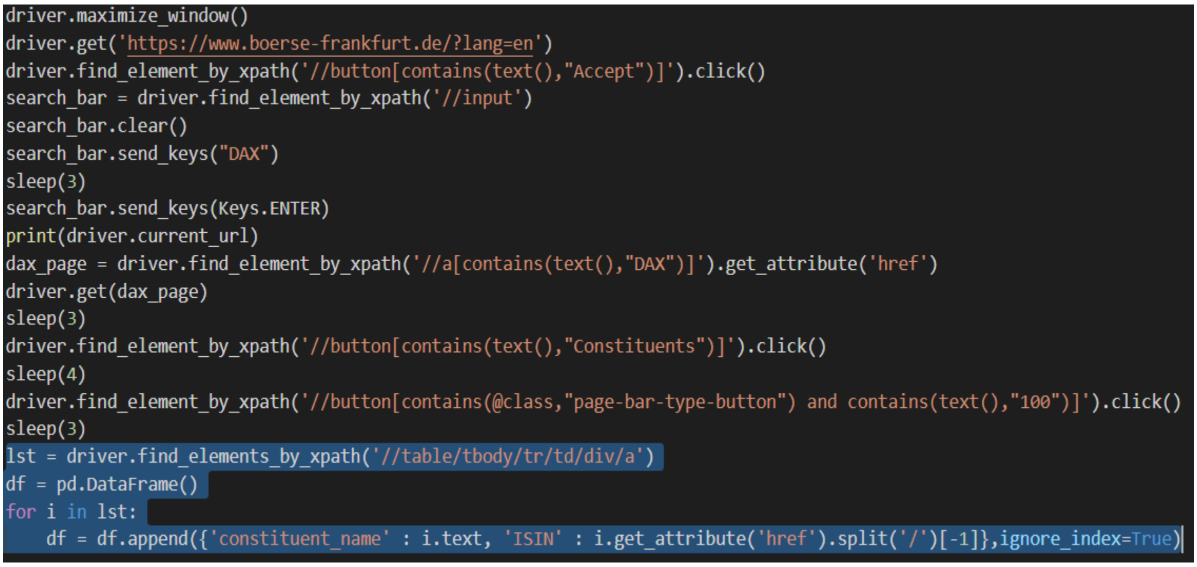
- **Scikit-learn**

  It is a free software machine learning library for the Python programming language. The framework is built on top of several popular Python packages, namely NumPy, SciPy, and matplotlib. Scikit-Learn is characterized by a clean, uniform, and streamlined API, as well as by very useful and complete online documentation. A benefit of this uniformity is that once you understand the basic use and syntax of Scikit-Learn for one type of model, switching to a new model or algorithm is very straightforward.

3.5 Configuration

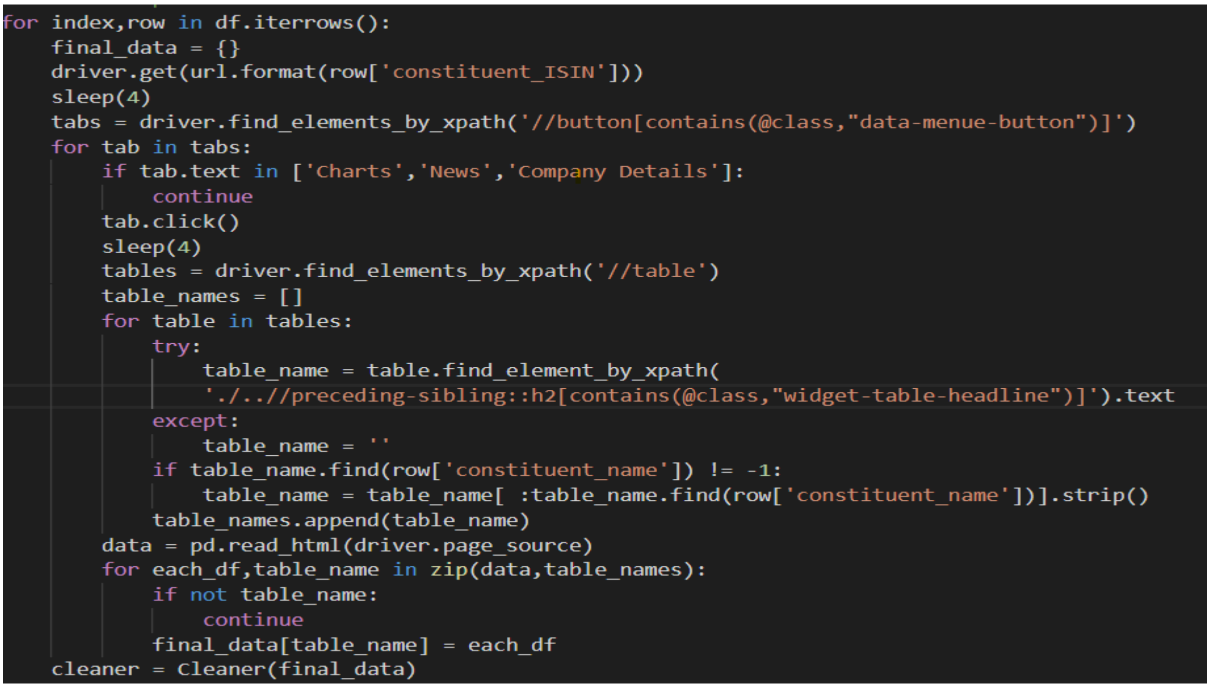
3.5.1 Scraping



```python
driver.maximize_window()
driver.get('https://www.boerse-frankfurt.de/?lang=en')
driver.find_element_by_xpath('//button[contains(text(),"Accept")]').click()
search_bar = driver.find_element_by_xpath('//input')
search_bar.clear()
search_bar.send_keys("DAX")
sleep(3)
search_bar.send_keys(Keys.ENTER)
print(driver.current_url)
dax_page = driver.find_element_by_xpath('//a[contains(text(),"DAX")]').get_attribute('href')
driver.get(dax_page)
sleep(3)
driver.find_element_by_xpath('//button[contains(text(),"Constituents")]').click()
sleep(4)
driver.find_element_by_xpath('//button[contains(@class,"page-bar-type-button") and contains(text(),"100")]').click()
sleep(3)
lst = driver.find_elements_by_xpath('//table/tbody/tr/td/div/a')
df = pd.DataFrame()
for i in lst:
    df = df.append({'constituent_name' : i.text, 'ISIN' : i.get_attribute('href').split('/')[-1]},ignore_index=True)
```

Figure 3.3 Collect all the constituents' name and WKN from website

The Figure 3.3 the code for collecting the constituents and its details provided by the website is collected. This data is stored in a sqlite3 database.

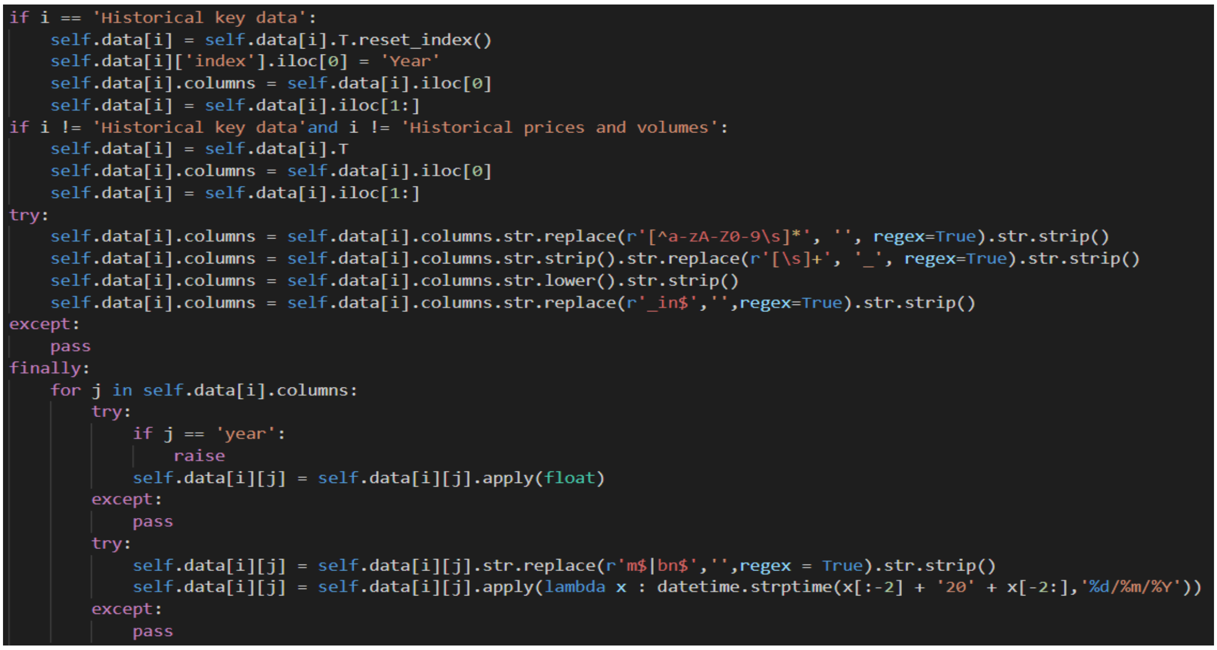


```
for index,row in df.iterrows():
    final_data = {}
    driver.get(url.format(row['constituent_ISIN']))
    sleep(4)
    tabs = driver.find_elements_by_xpath('//button[contains(@class,"data-menue-button")]')
    for tab in tabs:
        if tab.text in ['Charts','News','Company Details']:
            continue
        tab.click()
        sleep(4)
        tables = driver.find_elements_by_xpath('//table')
        table_names = []
        for table in tables:
            try:
                table_name = table.find_element_by_xpath(
                './../..//preceding-sibling::h2[contains(@class,"widget-table-headline")]').text
            except:
                table_name = ''
            if table_name.find(row['constituent_name']) != -1:
                table_name = table_name[ :table_name.find(row['constituent_name'])].strip()
            table_names.append(table_name)
        data = pd.read_html(driver.page_source)
        for each_df,table_name in zip(data,table_names):
            if not table_name:
                continue
            final_data[table_name] = each_df
    cleaner = Cleaner(final_data)
```

Figure 3.4 Collect stock related data for each constituent

The Figure 3.4 shows code snippet which collects stock related data from each constituent.

3.5.2 Cleaning

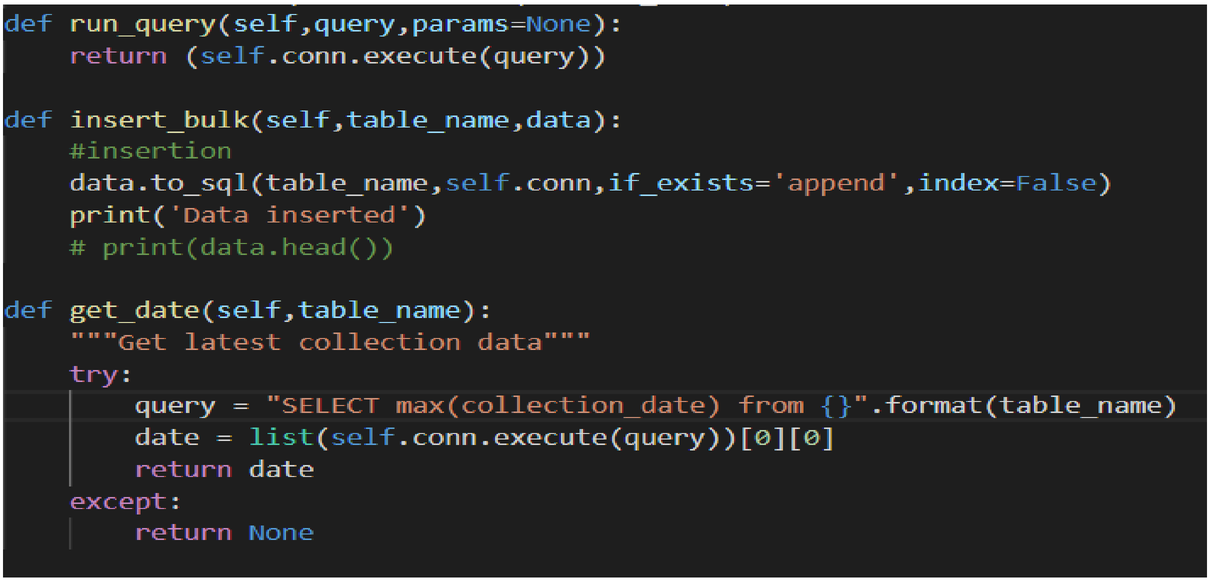


```
if i == 'Historical key data':
    self.data[i] = self.data[i].T.reset_index()
    self.data[i]['index'].iloc[0] = 'Year'
    self.data[i].columns = self.data[i].iloc[0]
    self.data[i] = self.data[i].iloc[1:]
if i != 'Historical key data'and i != 'Historical prices and volumes':
    self.data[i] = self.data[i].T
    self.data[i].columns = self.data[i].iloc[0]
    self.data[i] = self.data[i].iloc[1:]
try:
    self.data[i].columns = self.data[i].columns.str.replace(r'[^a-zA-Z0-9\s]*', '', regex=True).str.strip()
    self.data[i].columns = self.data[i].columns.str.strip().str.replace(r'[\s]+', '_', regex=True).str.strip()
    self.data[i].columns = self.data[i].columns.str.lower().str.strip()
    self.data[i].columns = self.data[i].columns.str.replace(r'_in$','',regex=True).str.strip()
except:
    pass
finally:
    for j in self.data[i].columns:
        try:
            if j == 'year':
                raise
            self.data[i][j] = self.data[i][j].apply(float)
        except:
            pass
        try:
            self.data[i][j] = self.data[i][j].str.replace(r'm$|bn$','',regex = True).str.strip()
            self.data[i][j] = self.data[i][j].apply(lambda x : datetime.strptime(x[:-2] + '20' + x[-2:],'%d/%m/%Y'))
        except:
            pass
```

Figure 3.5 Clean and pre-processing tabular data collected by collection script

Figure 5.5 shows code snippet to preprocess the tabular stock data collected. The data collected will not conform with the types in database, it may contain unwanted text characters which will is noise according to the model and hence these discrepancies will be removed.

3.5.3 Storing

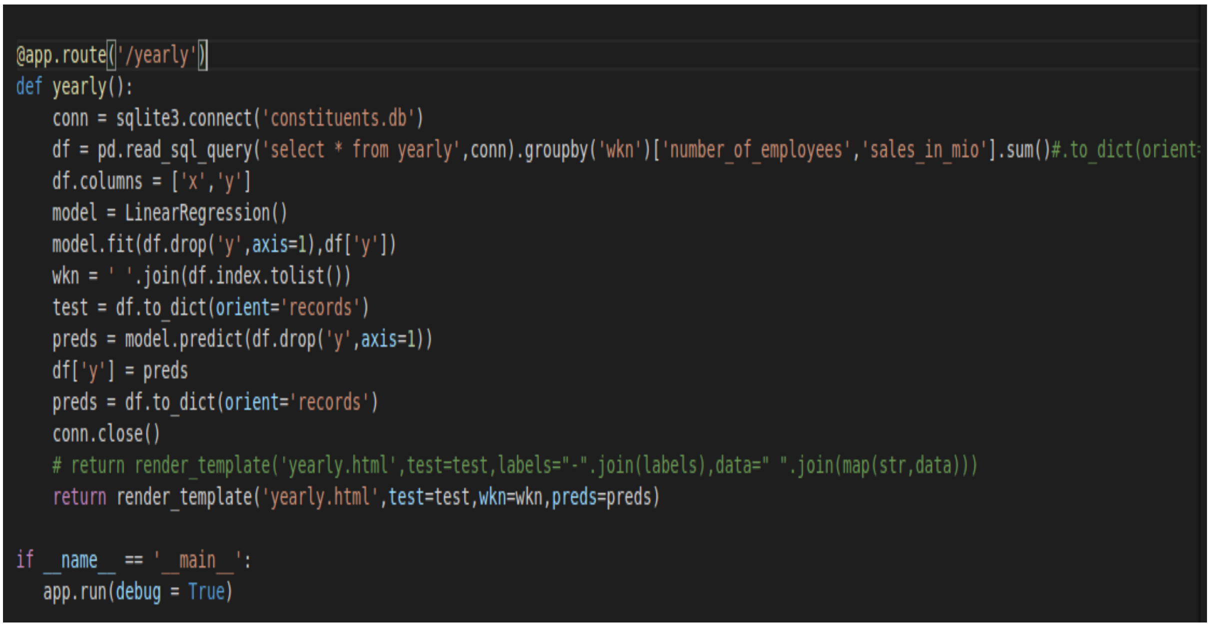


```
def run_query(self,query,params=None):
    return (self.conn.execute(query))

def insert_bulk(self,table_name,data):
    #insertion
    data.to_sql(table_name,self.conn,if_exists='append',index=False)
    print('Data inserted')
    # print(data.head())

def get_date(self,table_name):
    """Get latest collection data"""
    try:
        query = "SELECT max(collection_date) from {}".format(table_name)
        date = list(self.conn.execute(query))[0][0]
        return date
    except:
        return None
```

Figure 3.6 Storage utility which includes database function

The storage utility handles all data manipulation process on the database end. The code in Figure 3.6 shows the functions used for this manipulation which includes process like running a query to retrieve data and inserting data

3.5.4 Model



```
@app.route('/yearly')
def yearly():
    conn = sqlite3.connect('constituents.db')
    df = pd.read_sql_query('select * from yearly',conn).groupby('wkn')['number_of_employees','sales_in_mio'].sum()#.to_dict(orient=
    df.columns = ['x','y']
    model = LinearRegression()
    model.fit(df.drop('y',axis=1),df['y'])
    wkn = ' '.join(df.index.tolist())
    test = df.to_dict(orient='records')
    preds = model.predict(df.drop('y',axis=1))
    df['y'] = preds
    preds = df.to_dict(orient='records')
    conn.close()
    # return render_template('yearly.html',test=test,labels="-".join(labels),data=" ".join(map(str,data)))
    return render_template('yearly.html',test=test,wkn=wkn,preds=preds)

if __name__ == '__main__':
   app.run(debug = True)
```

Figure 3.7 Linear regression model

Figure 3.7 shows the code which predicts the stock trends using linear regression.

4.2 Results

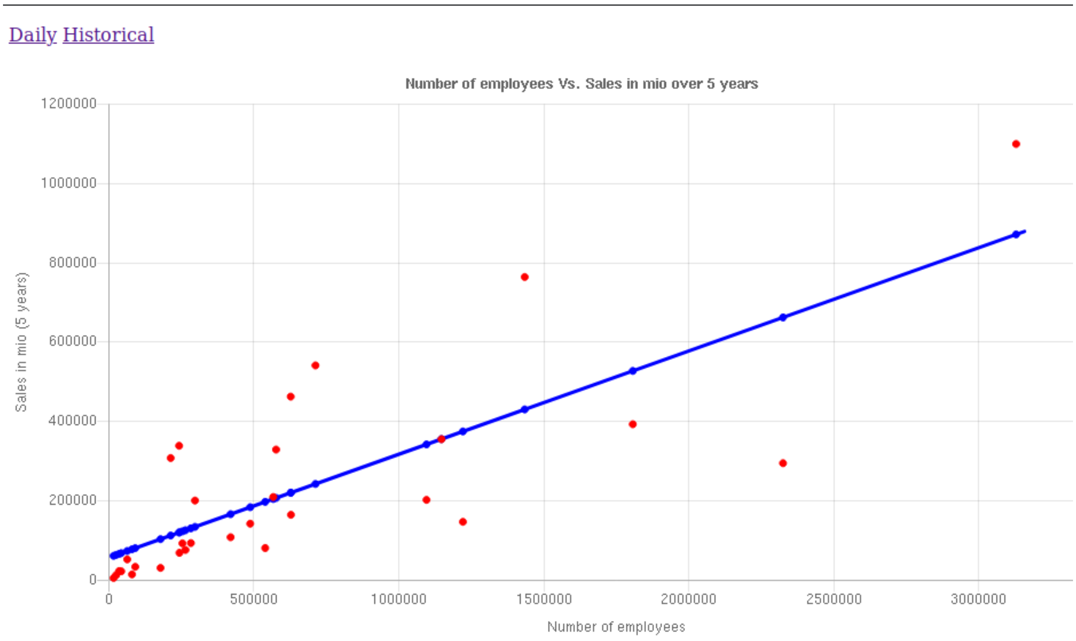


Figure 4.1 Yearly data trend

The figure 4.1 shows the linear model fitted for number of employees vs sales in million.

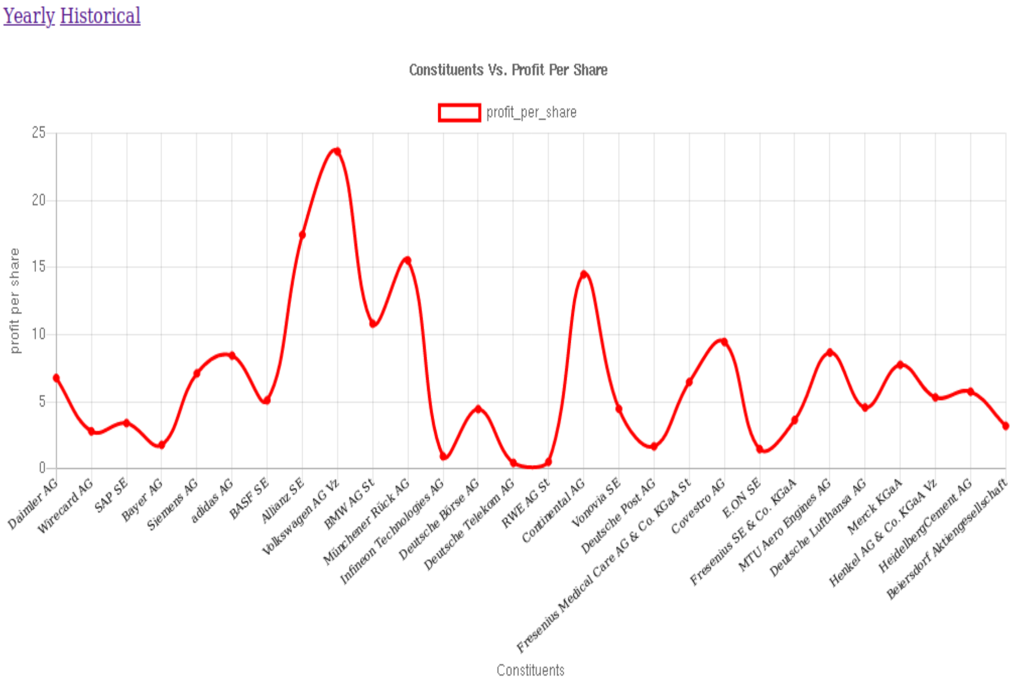


Figure 4.2 Daily data trend

Figure 4.2 shows profit per share for each constituent to help make decisions on which stock to buy or sell.

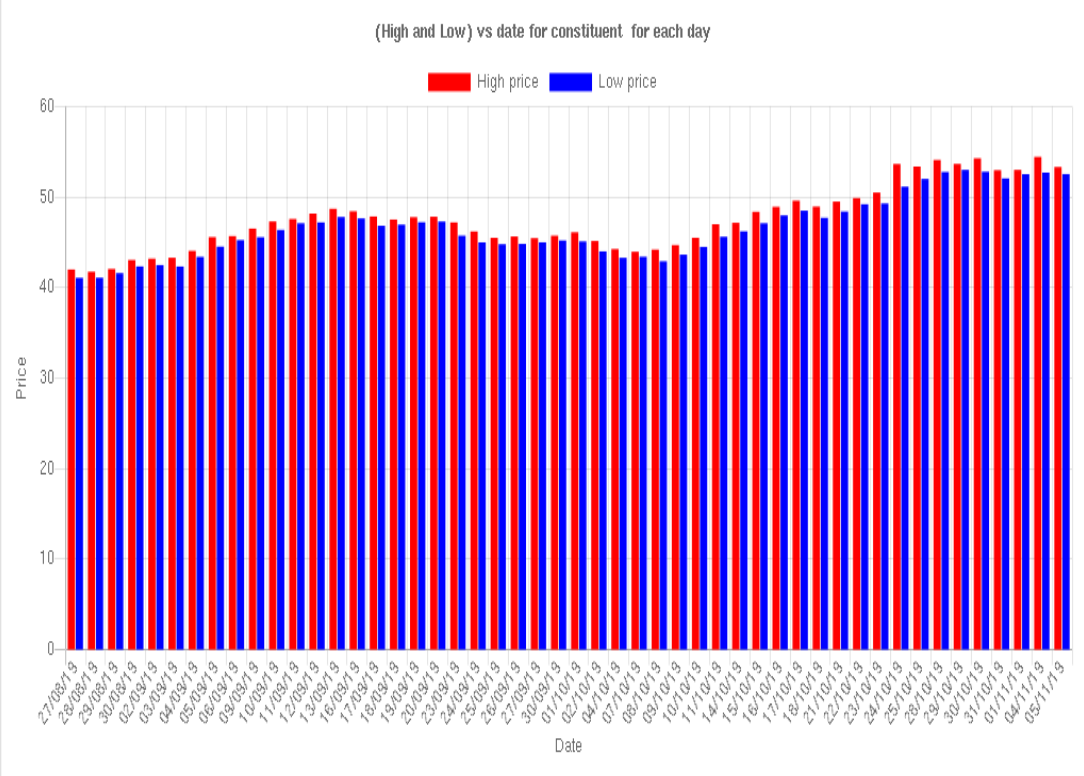
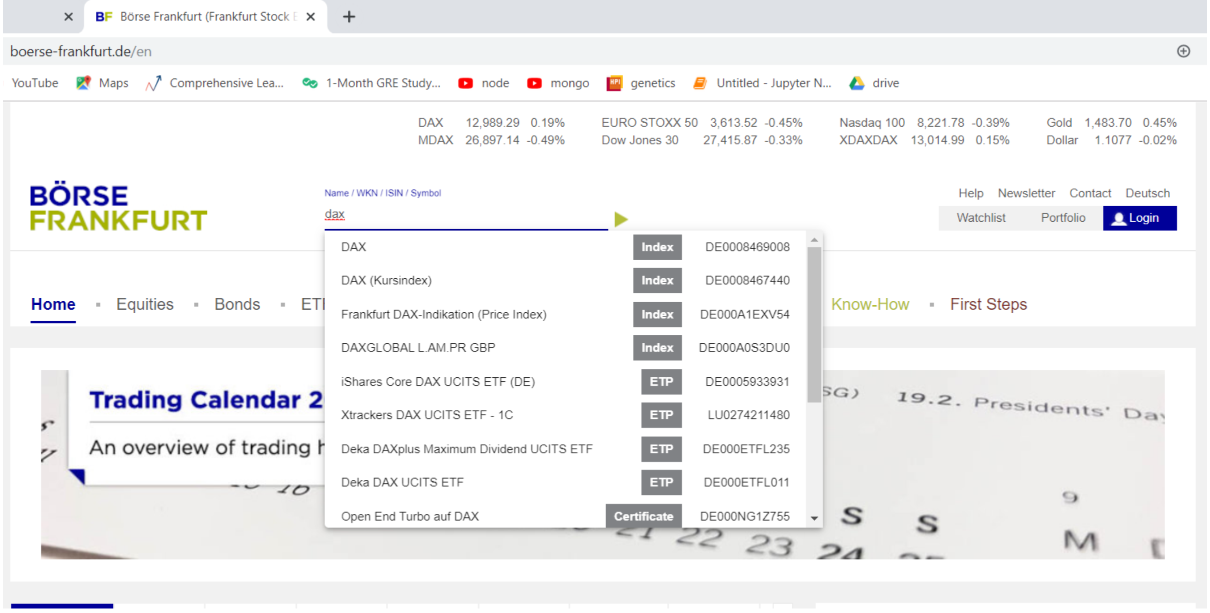


Figure 4.3 Historical data Bar chart

Figure 4.3 shows the high and low prices of the stock of a constituent on each day for the past month.

4.3 Snapshots



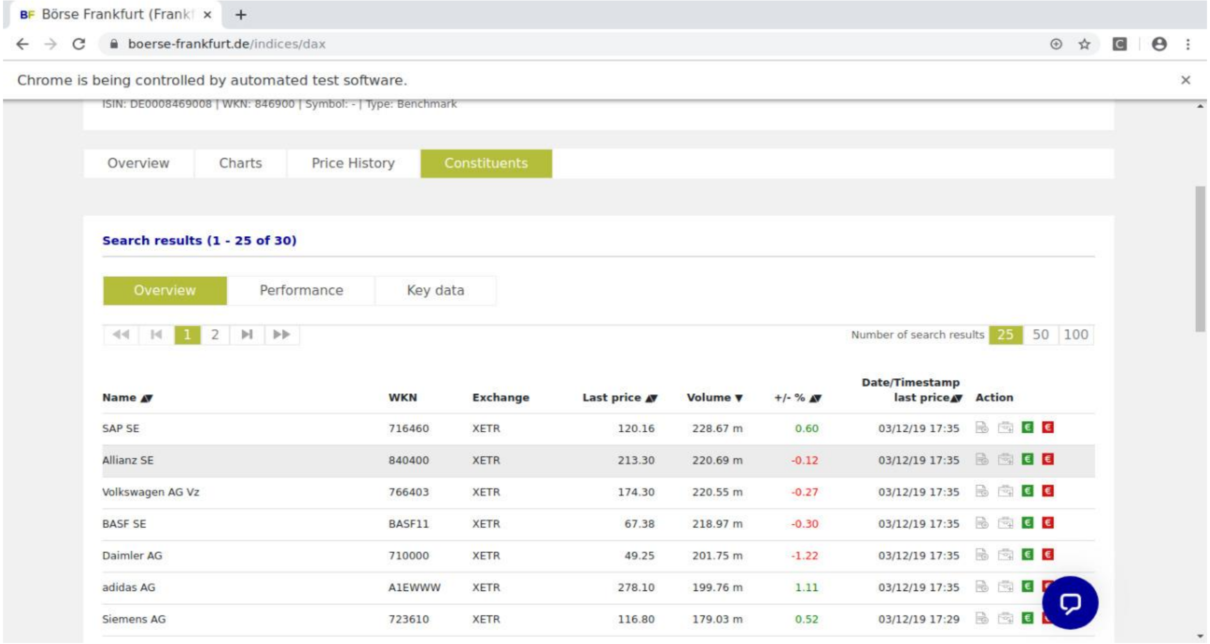
DAX 12,989.29 0.19%
MDAX 26,897.14 -0.49%
EURO STOXX 50 3,613.52 -0.45%
Dow Jones 30 27,415.87 -0.33%
Nasdaq 100 8,221.78 -0.39%
XDAXDAX 13,014.99 0.15%
Gold 1,483.70 0.45%
Dollar 1.1077 -0.02%

Name / WKN / ISIN / Symbol
dax

DAX	Index	DE0008469008
DAX (Kursindex)	Index	DE0008467440
Frankfurt DAX-Indikation (Price Index)	Index	DE000A1EXV54
DAXGLOBAL L.AM.PR GBP	Index	DE000A0S3DU0
iShares Core DAX UCITS ETF (DE)	ETP	DE0005933931
Xtrackers DAX UCITS ETF - 1C	ETP	LU0274211480
Deka DAXplus Maximum Dividend UCITS ETF	ETP	DE000ETFL235
Deka DAX UCITS ETF	ETP	DE000ETFL011
Open End Turbo auf DAX	Certificate	DE000NG1Z755

Figure 4.4 Sending keys to search box

The Figure 4.5 shows the selenium tool being used to enter keys into a search box.



Börse Frankfurt (Frankf ×
+
boerse-frankfurt.de/indices/dax
Overview
Charts
Price History
Constituents
Overview
Performance
Key data
◀◀ |◀ 1 2 ▶| ▶▶
Number of search results 25 50 100

Chrome is being controlled by automated test software.

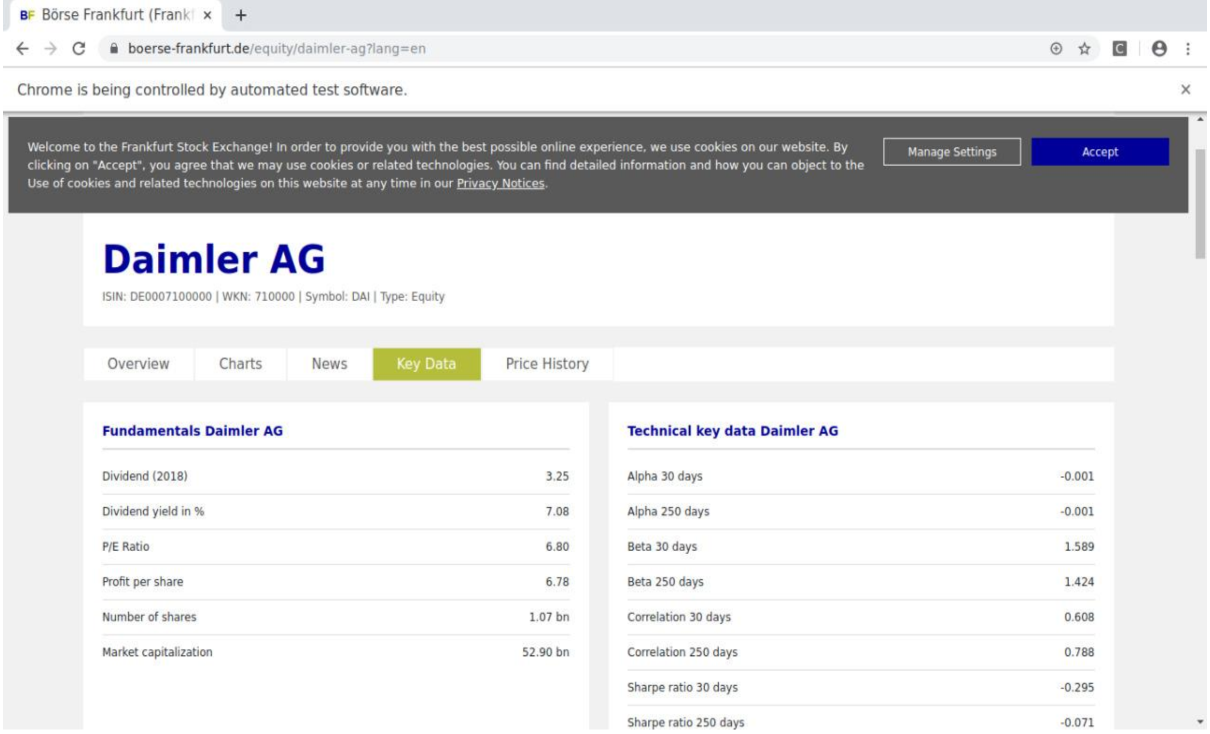
ISIN: DE0008469008 | WKN: 846900 | Symbol: - | Type: Benchmark

Search results (1 - 25 of 30)

Name	WKN	Exchange	Last price	Volume	+/- %	Date/Timestamp last price	Action
SAP SE	716460	XETR	120.16	228.67 m	0.60	03/12/19 17:35	€ €
Allianz SE	840400	XETR	213.30	220.69 m	-0.12	03/12/19 17:35	€ €
Volkswagen AG Vz	766403	XETR	174.30	220.55 m	-0.27	03/12/19 17:35	€ €
BASF SE	BASF11	XETR	67.38	218.97 m	-0.30	03/12/19 17:35	€ €
Daimler AG	710000	XETR	49.25	201.75 m	-1.22	03/12/19 17:35	€ €
adidas AG	A1EWWW	XETR	278.10	199.76 m	1.11	03/12/19 17:35	€
Siemens AG	723610	XETR	116.80	179.03 m	0.52	03/12/19 17:29	€

Figure 4.5 Collecting tabular data

The Figure 4.5 shows collecting of an entire table using the selenium tool



Börse Frankfurt (Frankf ×
+
boerse-frankfurt.de/equity/daimler-ag?lang=en
Manage Settings
Accept
Overview
Charts
News
Key Data
Price History

Chrome is being controlled by automated test software.

Welcome to the Frankfurt Stock Exchange! In order to provide you with the best possible online experience, we use cookies on our website. By clicking on "Accept", you agree that we may use cookies or related technologies. You can find detailed information and how you can object to the Use of cookies and related technologies on this website at any time in our Privacy Notices.

Daimler AG

ISIN: DE0007100000 | WKN: 710000 | Symbol: DAI | Type: Equity

Fundamentals Daimler AG	
Dividend (2018)	3.25
Dividend yield in %	7.08
P/E Ratio	6.80
Profit per share	6.78
Number of shares	1.07 bn
Market capitalization	52.90 bn

Technical key data Daimler AG	
Alpha 30 days	-0.001
Alpha 250 days	-0.001
Beta 30 days	1.589
Beta 250 days	1.424
Correlation 30 days	0.608
Correlation 250 days	0.788
Sharpe ratio 30 days	-0.295
Sharpe ratio 250 days	-0.071

Figure 4.6 Collecting multiple tables

The Figure 4.6 shows how the selenium tool collects multiple tables of different format and stores in a pandas dataframe.