In [872]:

```
import sys;
sys.path.insert(0, '..')
```

In [873]:

```
import findspark
findspark.init()
```

In [874]:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder. \
    appName("pyspark-1"). \
    enableHiveSupport(). \
    getOrCreate()
```

Read data

Sample function

In [1323]:

```
ort pyspark
ort re
m datetime import datetime
ort pyspark.sql.functions as sparkfunc
m pyspark.sql.types import DateType,IntegerType
ort pandas as pd
ort pyspark.sql.utils
m utils.distinct_values import get_distinct_values


is class is for csv file to be processed with the pyspark

ss CSVfiles:
initialisation, defining the class
ef __init__(self, file, path, header, data_type_changes_dict = {}):

 self.file = file
 self.path = path
 self.header = header
 self.data_type_changes_dict = data_type_changes_dict
 self.df = spark.read.csv(self.path +'/'+ self.file, header = self.header)
method to wrok with the unprocessed csv file (raw data layer)
ef unsparked(self):

 df = spark.read.csv(self.path +'/'+ self.file, header = self.header)
 return df
method to wrok with the processed csv file (operational and analytical data layers)
ef sparked(self):
 #putting the csv file data into spark dataframe
 df = self.unsparked()
 #checking if any changes to be performed on data have been passed
 if self.data_type_changes_dict == {}:
     return df
 else:
     #get the list of columns' names of the dataframe
     header_list = self.unsparked().columns[:]
     #to fix the names of the headers for pyspark standards (restricted symbols) and
     for i in range(len(header_list)):
     #get the initial name of the field
         column_name = header_list[i]
     #removes any non-numeric OR non-letter symbol in a column name into _ and lowers
         header_list[i] =  re.sub('[^a-zA-Z0-9] *','_',header_list[i]).lower()
     #roam through the datachange dict to verify if the dataset field needs to have c
         for k,v in self.data_type_changes_dict.items():
             if k in header_list[i].split('_'):
                 #changing the format
                 df = df.withColumn(column_name, sparkfunc.col(column_name).cast(v))
     #renaming the columns of the processed dataset
     df=df.toDF(*header_list)
     #returning the processed dataframe
     return df
methods to explore the csv file
names of the methods are reflecting the purpose of it
ef column_data_type(self, column_name):
 return str(self.sparked().schema).split(column_name)[1].split(',')[1]

ef num_of_nulls(self, column_name):
 #verifying the nulls
 return self.sparked().filter(sparkfunc.col(column_name).isNull()).count()
```

```
ef num_of_records(self, column_name = ''):
 return self.sparked().count()

ef max_length(self, column_name):
 return self.sparked().select(sparkfunc.length(column_name)).groupby().max().collect(

ef min_length(self, column_name):
 return self.sparked().select(sparkfunc.length(column_name)).groupby().min().collect(
min and max values are available for non-string data as it can be quite large
ef max_value(self, column_name):
 if self.column_data_type(column_name) != 'StringType':
     if self.column_data_type(column_name) == 'DateType':
         return self.sparked().select(sparkfunc.max(column_name)).collect()[0][0].str
     else:
         return self.sparked().select(sparkfunc.max(column_name)).collect()[0][0]
 else:
     return ''

ef min_value(self, column_name):
 if self.column_data_type(column_name) != 'StringType':
     if self.column_data_type(column_name) == 'DateType':
         return self.sparked().select(sparkfunc.min(column_name)).collect()[0][0].str
     else:
         return self.sparked().select(sparkfunc.min(column_name)).collect()[0][0]
 else:
     return ''
using spark built-in function
ef distinct_values_count(self, column_name):
 return len(get_distinct_values(self.sparked(), column_name))

method, which uses the class methods to describe the data of the csv file, based on t
which fields he/she is interested in and what metrics he/she is interested in for tho
ef metadata_info(self, metrics_list, columns_list):
     #defining the calss in order to refer to it defined not like an object, but like
     workingclass = globals()['CSVfiles'](
                                 file = self.file
                                 ,path = self.path
                                 ,header = self.header
                                 ,data_type_changes_dict = self.data_type_changes_dic
                                 )
     #the output of this method
     explain_metadata_extended_schema = ["field","metrics"]
     #creating the lambda function dynamically: in accordance to what information use
     def presentation_parameters(structure_schema_list, metrics_list):
         lfunc = 'lambda x: (x.'+ structure_schema_list[0] +','
         lfunc += ' , '.join(['x.' + structure_schema_list[1] + '["' + metrics_list[i
         lfunc += ')'
         return lfunc
     #defining the output dataframe to be loaded with the calculated data as a list
     explain_metadata_extended = []

     for c in columns_list:
         column_metric_value_dict_list = [] #defifning the {metric:value} list of dic
         column_metric_value_dict_list.append(c) #appending iterated column name to i
         metric_value_dict = {}#defining the {metric:value} dictionary
         for m in metrics_list:
             mfunc = getattr(workingclass, m) #calling the class function, defining i
             metric_value_dict[m] = mfunc(c) #adding the metric-value pair to the dic

         column_metric_value_dict_list.append(metric_value_dict) #adding all the usei
```

```
        explain_metadata_extended.append(tuple(column_metric_value_dict_list)) #appe
    #forming the spark dataframe
    df = spark.createDataFrame(data=explain_metadata_extended, schema = explain_meta
    #forming the structure of the dataframe as per the metrics to be displayed: name
    map_structure = presentation_parameters(explain_metadata_extended_schema, metric
    #adding the name of the first column in the output dataframe
    metrics_list.insert(0,explain_metadata_extended_schema[0])
    #returning the dataframe with the name of the clumns of the file and the metrics
    return df.rdd.map(eval(map_structure)).toDF(metrics_list)
```

In [1324]:

```
#create a dict where keys are words in header names, which point to datatype of the
#formats to work with are mentioned in the import section
datachange = {"date":"date","until":"date","updated":"date","id":"int", "range":"int
#defining the file, to use created class
f = CSVfiles(
             file = 'nyc-jobs.csv'
             ,path = '/dataset'
             ,header = True
             ,data_type_changes_dict = datachange
            )
#user-defined columns of the dataset to be assesed with the metrics
columns_to_asses = f.sparked().columns[:]
#defining those metrics, which meed to asses the file
metrics_to_display = ['column_data_type','num_of_records','num_of_nulls','max_length
#calling the method
df2 = f.metadata_info(metrics_to_display, columns_to_asses)
#visualizing as grid (the more fileds and metris - the more time it is taking for ca
df2.toPandas()
```

Out[1324]:

	field	column_data_type	num_of_records	num_of_nulls	max_length	mi
0	job_id	IntegerType	2946	0	6	
1	agency	StringType	2946	0	30	
2	posting_type	StringType	2946	0	8	
3	_of_positions	StringType	2946	0	3	
4	business_title	StringType	2946	0	117	
5	civil_service_title	StringType	2946	0	30	
6	title_code_no	StringType	2946	0	5	
7	level	StringType	2946	0	2	
8	job_category	StringType	2946	2	201	
9	full_time_part_time_indicator	StringType	2946	195	1	
10	salary_range_from	IntegerType	2946	0	6	
11	salary_range_to	IntegerType	2946	0	6	
12	salary_frequency	StringType	2946	0	6	
13	work_location	StringType	2946	0	30	
14	division_work_unit	StringType	2946	0	30	
15	job_description	StringType	2946	0	10699	
16	minimum_qual_requirements	StringType	2946	18	2791	
17	preferred_skills	StringType	2946	259	2852	
18	additional_information	StringType	2946	563	2148	
19	to_apply	StringType	2946	180	2508	
20	hours_shift	StringType	2946	1062	2449	
21	work_location_1	StringType	2946	1138	2062	

	field	column_data_type	num_of_records	num_of_nulls	max_length	mi
22	recruitment_contact	StringType	2946	1763	2807	
23	residency_requirement	StringType	2946	678	1975	
24	posting_date	DateType	2946	1566	10	
25	post_until	DateType	2946	2212	10	
26	posting_updated	DateType	2946	1188	10	
27	process_date	DateType	2946	927	10	

In [ ]:

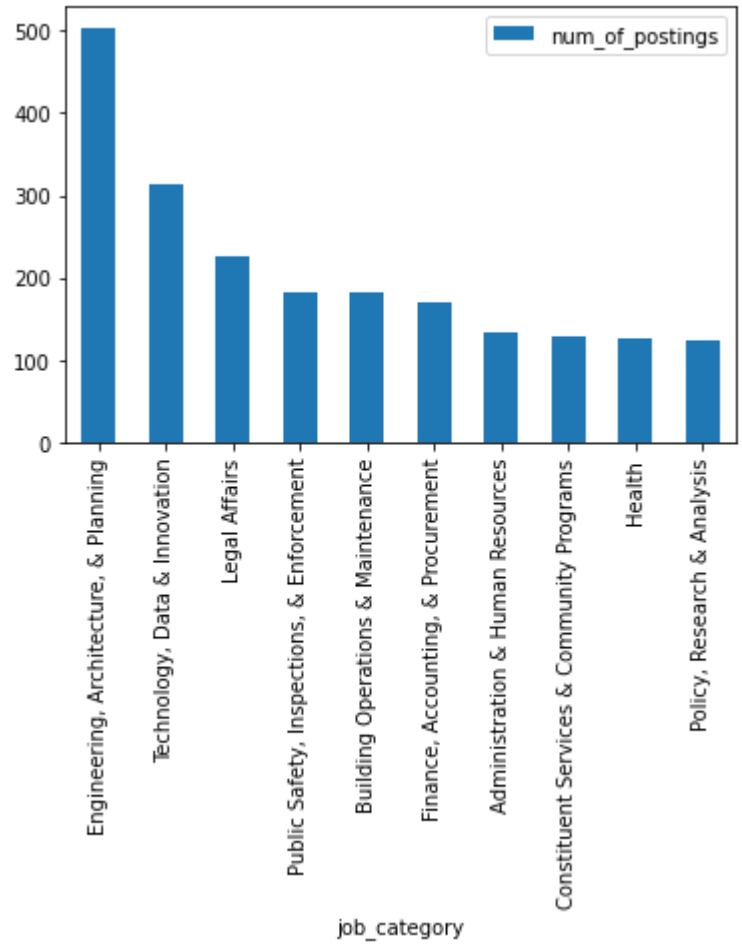
```
What's the number of jobs posting per category (Top 10)?
```

In [1211]:

```
#job_id is not the unique identifier as a posting can be internal or external

#that is why distinct is not used

#calculating number of postings per each category
ch1 = f.sparked().groupBy("job_category").agg(sparkfunc.count(sparkfunc.col("job_id"
#limiting it to the top 10 (by limiting the output ordered in desc way) and displayi
ch1.orderBy("num_of_postings", ascending=False).limit(10).toPandas().plot.bar(x = 'j
```

Out[1211]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc1e85b2dd8>
```



In [ ]:

What's the salary distribution per job category?

In [1212]:

```
#the number of job categories is high, so, it is used as input parameter
#taking the category, having the most number of postings
jc = 'Engineering, Architecture, & Planning'
#defining, which type of salary (there are 2 - minimum and maximun in the fork) to u
salary_data_type = 'salary_range_from'
#calculating, how many times the mentioned salary is within the postings
salary_distribution_stg = f.sparked().filter(sparkfunc.col("job_category")==jc).grou


#showing the results as grid
salary_distribution_stg.orderBy("salary_distribution", salary_data_type,ascending=Fa
```

Out[1212]:

	job_category	salary_range_from	salary_distribution
0	Engineering, Architecture, & Planning	55416	38
1	Engineering, Architecture, & Planning	65783	27
2	Engineering, Architecture, & Planning	57078	25
3	Engineering, Architecture, & Planning	78210	22
4	Engineering, Architecture, & Planning	74990	22
...	...	...	...
73	Engineering, Architecture, & Planning	37796	2
74	Engineering, Architecture, & Planning	363	2
75	Engineering, Architecture, & Planning	45	2
76	Engineering, Architecture, & Planning	91616	1
77	Engineering, Architecture, & Planning	53	1

78 rows × 3 columns

In [1213]:

```
#there are many ways to analyze the distribution
#here:

#per each frequency calculate the min, max and/or average salary

#average salary needs a specific weighted calculation, which is shown in the next ch

#for this challenge the min_salary is used
ordering_field = "min_salary"
#calculation
ch2 = salary_distribution_stg.groupBy("salary_distribution").agg(sparkfunc.min(spark
                                                    sparkfunc.max(sparkfunc.col("salary_ran
#displaying the calculated data
ch2.orderBy(ordering_field).toPandas()

#number of cases can be recalculated as divided by total number, however,
#for this challenge it won't make a difference
```

Out[1213]:

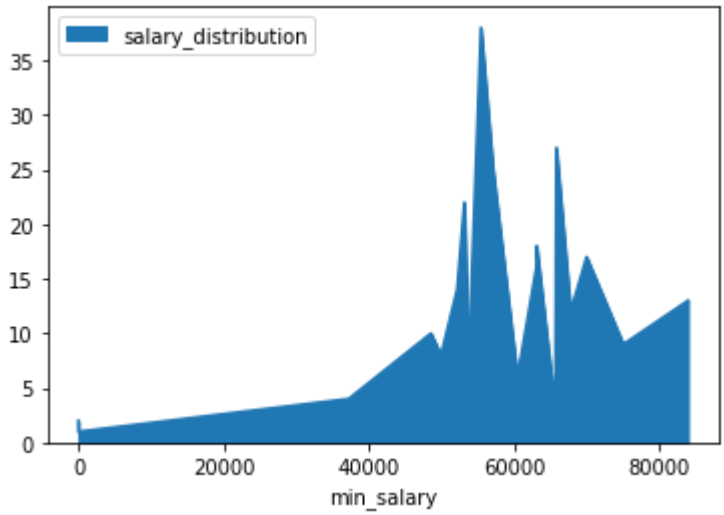
	salary_distribution	min_salary	max_salary
0	2	45	140000
1	1	53	91616
2	4	37217	110000
3	10	48535	57720
4	8	49916	87490
5	14	52137	52137
6	22	53134	78210
7	7	53702	90114
8	38	55416	55416
9	25	57078	57078
10	6	60435	72038
11	16	63031	63031
12	18	63074	63074
13	3	65640	65640
14	27	65783	65783
15	12	67757	70000
16	17	69940	69940
17	9	75000	80557
18	13	83887	83887

In [1214]:

```
#displaying the calculation as an area graph
ch2.sort(ordering_field).toPandas().plot.area(x = ordering_field, y = "salary_distri
```

Out[1214]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc1e85a7828>
```



In [ ]:

```
Is there any correlation between the higher degree and the salary?
```

In [1215]:

```
from utils.distinct_values import get_distinct_values
#calculating the distinct values of the level column
levels_list = get_distinct_values(df = f.sparked(), column = 'level')
#it is clear, that different agencies are using different gradations of the level
#in order to calculate the correlation those gradations have to be categorized
#per each gradation category the correlation has to be calculated
#here is an example of possible categorization, using lambda function
categorization = lambda x: "group1" if len(x)==1 else ("group2" if isinstance(x[:1]
#creating list of categories for the list of levels
levels_list_cat = list(map(categorization, levels_list))
#creating list of tuples with the format (level, category)
categorized_levels = list(map(lambda x,y: tuple([x,y]), levels_list, levels_list_cat
#making spark dataframe out of it
level_categorization = spark.createDataFrame(data = categorized_levels, schema = ["l
lc = level_categorization
#displaying it as a grid
lc.toPandas()
```

Out[1215]:

	level_c	category
0	3	group1
1	M4	group2
2	M7	group2
3	4B	group2
4	0	group1
5	M6	group2
6	M1	group2
7	4A	group2
8	M5	group2
9	M2	group2
10	1	group1
11	M3	group2
12	4	group1
13	2	group1

In [1216]:

```
#importing component for the windows functions
from pyspark.sql.window import Window
#per each level number of postings with the min/max/avg salary is calculated
#salary for calculation is used from challenge 2
ch2_stg = f.sparked().groupBy("level").agg(sparkfunc.count(sparkfunc.col("job_id")).
                                           sparkfunc.min(salary_data_type).alias("min_sa
                                           sparkfunc.max(salary_data_type).alias("max_sa
                                           sparkfunc.avg(salary_data_type).alias("avg_sa

#the number of all postings have to be calcuated within a level categorization
#in order to calculate the weighted statistics for calculation of the estimation

#the average salary used in correlation calculation will be multiplied with the weig

#secondly, as levels are not all integers, the ranking is introduced within each sor
#because correlation function can only use 2 numeric rows
windowSpec  = Window.partitionBy("category")
windowSpec2  = Window.partitionBy("category").orderBy("level")

ch2_stg3 = ch2_stg.join(lc, ch2_stg["level"] == lc["level_c"],'left') \
            .withColumn("num_of_postings_weighted",sparkfunc.col("num_of_postings")/
            .withColumn("level_new",sparkfunc.rank().over(windowSpec2)) \
            .sort("level")


ch2_stg4 = ch2_stg3.filter(sparkfunc.col("category")=='group1') \
            .withColumn("avg_salary_corrected",sparkfunc.col("avg_salary")*sparkfunc

ch2_stg4.toPandas()
```

Out[1216]:

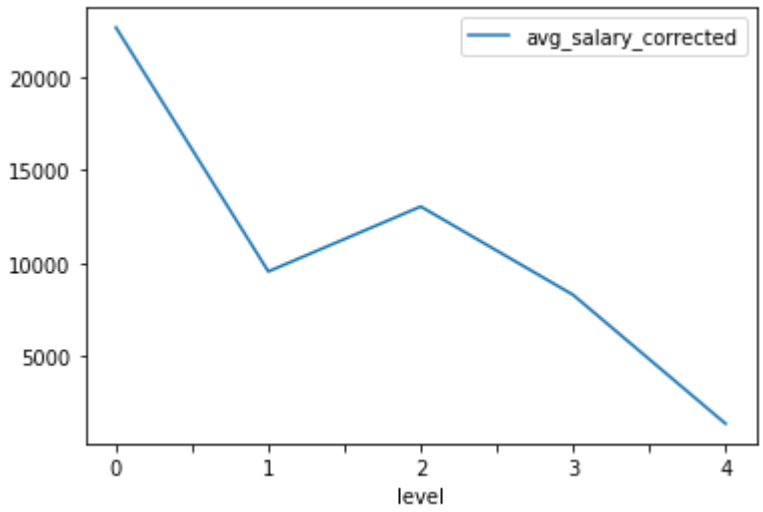
	level	num_of_postings	min_salary	max_salary	avg_salary	level_c	category	num_of_post
0	0	1112	0	153666	50568.967626	0	group1	
1	1	521	8	85000	45510.243762	1	group1	
2	2	505	0	98388	64100.122772	2	group1	
3	3	299	16	105000	68888.157191	3	group1	
4	4	47	37251	157725	72950.659574	4	group1	

In [1217]:

```
#calculation of the correlation in between the corrected average salary
#one can say by the coefficient value (89% of the cases) that it's not rejected, tha
#negative value shows the down trend
ch2_stg4.sort("level").toPandas().plot.line(x = "level", y = "avg_salary_corrected")
ch2_stg4.corr("level_new", "avg_salary_corrected")
```

Out[1217]:

```
-0.8905448738018038
```

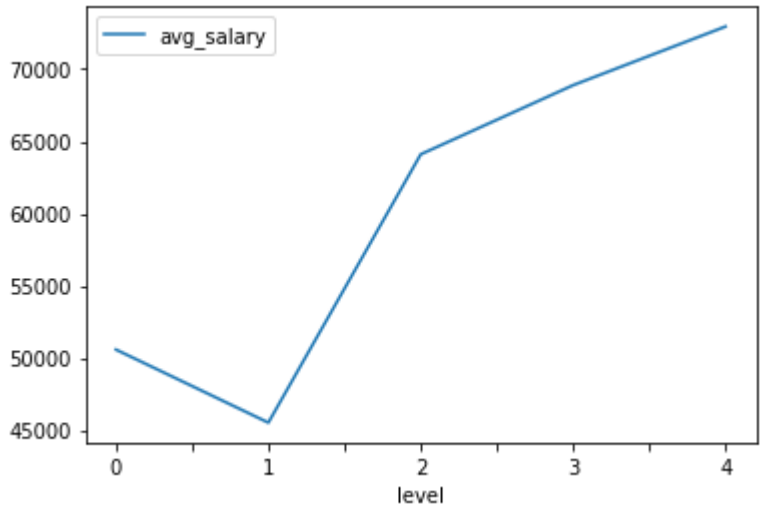


In [1219]:

```
#calculation of the correlation in between the NOT corrected average salary

#one can say by the coefficient value (90% of the cases) that it's not rejected, tha
#positive value shows the up trend

#the correction of the average salary by number of observations makes a totqlly oppo
ch2_stg4.sort("level").toPandas().plot.line(x = "level", y = "avg_salary")
ch2_stg4.corr("level_new", "avg_salary")
```

Out[1219]:

```
0.909268023999053
```

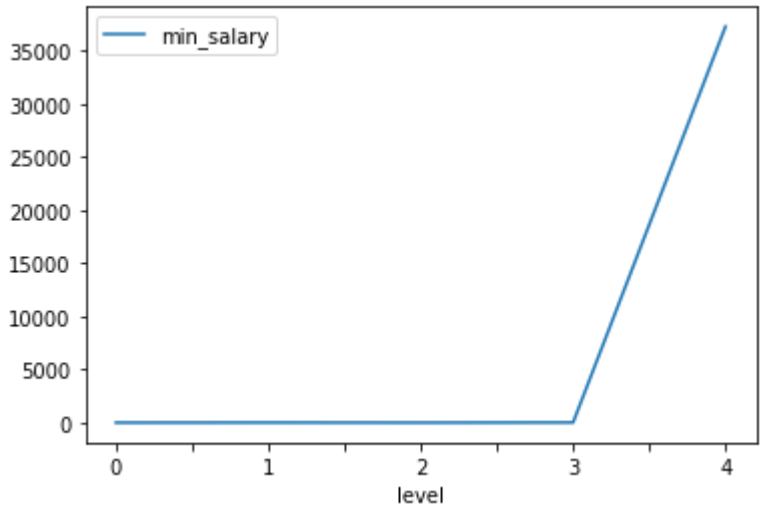


In [1218]:

```
#calculation of the correlation in between the min salary

#one can say by the coefficient value (70% 0f the cases) that it's not rejected, tha
#positive value shows the up trend

#in this case more thorough investigation is needed

ch2_stg4.sort("level").toPandas().plot.line(x = "level", y = "min_salary")
ch2_stg4.corr("level_new", "min_salary")
```

Out[1218]:

```
0.7072965779074636
```

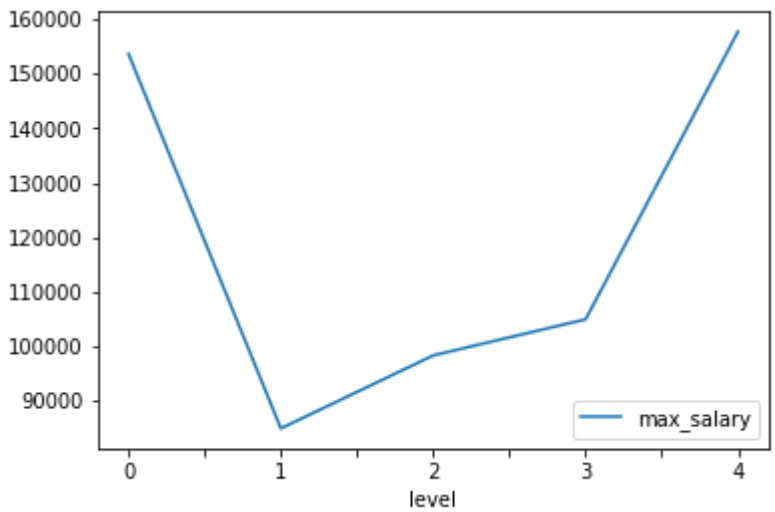


In [1058]:

```
#calculation of the correlation in between the max salary

#one can say by the coefficient value that it's not accepted, that correlation is th
#the trend is not clear by the graph, though the coefficient is positive

ch2_stg4.sort("level").toPandas().plot.line(x = "level", y = "max_salary")
ch2_stg4.corr("level_new", "max_salary")
```

Out[1058]:

```
0.13293940115218222
```



In [ ]:

```
What's the job posting, having the highest salary per agency?
```

In [1148]:

```
#salary type is taken as in previous challenge and challenge 2 (salary_data_type)
# window function to calculate the maximun salary across all agency postings
windowSpec = Window.partitionBy("agency")

ch4_stg = f.sparked() \
            .withColumn("agency_max_salary", sparkfunc.max(salary_data_type).over(wi
#filtering the postings, having the maximum salary in the agency
ch4 = ch4_stg.filter(sparkfunc.col(salary_data_type) == sparkfunc.col("agency_max_sa
ch4.toPandas()
```

Out[1148]:

	job_id	agency	posting_type	_of_positions	business_title	civil_service_title
0	425347	LANDMARKS PRESERVATION COMM	Internal	1	LANDMARKS PRESERVATIONIST, PRESERVATION DEPT	LANDMARKS PRESERVATIONIST
1	425347	LANDMARKS PRESERVATION COMM	External	1	LANDMARKS PRESERVATIONIST, PRESERVATION DEPT	LANDMARKS PRESERVATIONIST
2	170989	OFFICE OF COLLECTIVE BARGAININ	Internal	1	COLLEGE AIDE - CLERICAL	COLLEGE AIDE (ALL CITY DEPTS)
3	170989	OFFICE OF COLLECTIVE BARGAININ	External	1	COLLEGE AIDE - CLERICAL	COLLEGE AIDE (ALL CITY DEPTS)
4	415570	FIRE DEPARTMENT	Internal	1	Clinical Director for the Couseling Services Unit	ADMINISTRATIVE PSYCHOLOGIST
...	...	...	...	...	...	...
111	413287	OFF OF PAYROLL ADMINISTRATION	Internal	1	Fiscal Analyst	STAFF ANALYST
112	416542	NYC HOUSING AUTHORITY	External	1	Vice-President for Support Services	ADMINISTRATIVE STAFF ANALYST (
113	416542	NYC HOUSING AUTHORITY	Internal	1	Vice-President for Support Services	ADMINISTRATIVE STAFF ANALYST (
114	423103	DEPARTMENT OF INVESTIGATION	Internal	1	Deputy Commissioner for Operations	DEPUTY COMMISSIONER

	job_id	agency	posting_type	_of_positions	business_title	civil_service_title
115	423103	DEPARTMENT OF INVESTIGATION	External	1	Deputy Commissioner for Operations	DEPUTY COMMISSIONER

116 rows × 29 columns

In [ ]:

```
What's the job postings average salary per agency for the last 2 years?
```

In [1142]:

```
from datetime import datetime
from dateutil.relativedelta import relativedelta
#calculating the date, where the last 2 years of data sample starts, based on the da
diff = datetime.strptime(f.max_value('posting_date'), '%Y-%m-%d') - relativedelta(ye
#filtering the data by the date calculated
#grouping by the agency, aggregating by the average salary
ch5_stg = f.sparked().filter(sparkfunc.col("posting_date") > '2019-08-26') \
            .groupBy("agency").agg(sparkfunc.avg(sparkfunc.col(salary_data_type)).al
#showing the data as grid without rounding
ch5_stg.toPandas()
```

Out[1142]:

	agency	avg_salary_2years
0	FIRE DEPARTMENT	47113.500000
1	ADMIN FOR CHILDREN'S SVCS	53686.000000
2	TAX COMMISSION	16432.500000
3	HRA/DEPT OF SOCIAL SERVICES	54709.444444
4	TAXI & LIMOUSINE COMMISSION	48616.500000
5	DEPARTMENT OF BUSINESS SERV.	51548.500000
6	DEPT OF DESIGN & CONSTRUCTION	61571.578947
7	TEACHERS RETIREMENT SYSTEM	62397.000000
8	FINANCIAL INFO SVCS AGENCY	81992.600000
9	DEPARTMENT OF CORRECTION	27382.500000
10	HOUSING PRESERVATION & DVLPMNT	82190.307692
11	CIVILIAN COMPLAINT REVIEW BD	47103.000000
12	OFFICE OF MANAGEMENT & BUDGET	66092.458333
13	MAYORS OFFICE OF CONTRACT SVCS	75357.142857
14	DEPT OF CITYWIDE ADMIN SVCS	28444.666667
15	ADMIN TRIALS AND HEARINGS	23619.260870
16	DEPARTMENT OF SANITATION	19145.500000
17	DEPT. OF HOMELESS SERVICES	31573.000000
18	DEPT OF HEALTH/MENTAL HYGIENE	40896.938776
19	POLICE DEPARTMENT	56872.611111
20	HUMAN RIGHTS COMMISSION	43173.750000
21	DISTRICT ATTORNEY RICHMOND COU	63333.333333
22	PRESIDENT BOROUGH OF MANHATTAN	50000.000000
23	DEPARTMENT OF BUILDINGS	18248.750000
24	DEPARTMENT OF TRANSPORTATION	45468.758621
25	NYC EMPLOYEES RETIREMENT SYS	45688.666667
26	LAW DEPARTMENT	76904.600000

	agency	avg_salary_2years
27	DEPT OF INFO TECH & TELECOMM	61306.204545
28	CONFLICTS OF INTEREST BOARD	100000.000000
29	OFFICE OF THE COMPTROLLER	72666.666667
30	DEPARTMENT OF PROBATION	34676.333333
31	DISTRICT ATTORNEY KINGS COUNTY	64813.166667
32	DEPT OF YOUTH & COMM DEV SRVS	49643.000000
33	DEPT OF ENVIRONMENT PROTECTION	60435.703704
34	CONSUMER AFFAIRS	69194.857143
35	BOROUGH PRESIDENT-QUEENS	37217.000000
36	DEPT OF PARKS & RECREATION	58134.000000
37	DEPARTMENT FOR THE AGING	38909.500000
38	OFF OF PAYROLL ADMINISTRATION	45139.714286
39	NYC HOUSING AUTHORITY	55531.096774
40	DEPARTMENT OF INVESTIGATION	50760.000000

In [ ]:

```
What are the highest paid skills in the US market?
```

In [1208]:

```
ting the data from challenge 4, where the highest salaried per agency were calculate
tering the preferred skills columns for distinct values only
stg = ch4.select(sparkfunc.col("preferred_skills")).distinct()
oving all non-letter symbols (not removing blanks)
stg1 = ch6_stg.withColumn("skills_clean", sparkfunc.regexp_replace(sparkfunc.col("pre
ing a dataframe of "opened" - exploded - lists of the words from preferrred_skills
ing those distinct and calculating number of times, mentioned within preffered_skills
stg2 = ch6_stg1.withColumn('skills', sparkfunc.explode(sparkfunc.split(sparkfunc.lowe
istinct() \
roupBy('skills') \
ount()
tering out all the words, which length is less than 4 letters
wing a word from preffered skills with the number of times mentioned
stg2.filter(sparkfunc.length(sparkfunc.col("skills")) > 4).sort('count', 'skills', as
```

Out[1208]:

	skills	count
0	experience	29
1	skills	24
2	strong	20
3	excellent	18
4	ability	18
5	communication	17
6	years	14
7	written	14
8	analytical	14
9	knowledge	12
10	organizational	11
11	interpersonal	11
12	including	11
13	candidates	11
14	working	10
15	preferred	10
16	microsoft	10
17	management	10
18	excel	9
19	development	9
20	writing	8
21	service	8
22	research	8
23	least	8

	skills	count
24	detail	8
25	degree	8
26	verbal	7
27	tools	7
28	required	7
29	office	7
30	demonstrated	7
31	deadlines	7
32	computer	7
33	staff	6
34	should	6
35	proficiency	6
36	multiple	6
37	legal	6
38	government	6
39	familiarity	6
40	environment	6
41	complex	6
42	candidate	6
43	andor	6
44	above	6
45	which	5
46	understanding	5
47	rules	5
48	related	5
49	public	5

In [1442]:

```
#testing challenge
#function to run tests for all the methods in the class defined

def test_functions(assesment_file, af_path, af_header, af_data_type_changes_dict, te

    ff = CSVfiles(
              file = 'nyc-jobs.csv'
             ,path = '/dataset'
             ,header = True
             ,data_type_changes_dict = datachange
            )

    mock_df_schema: list = ['field','metrics']

    metrics_list = []
    columns_list = []
    results = [] #array, which will be containing the results of tests

    for el in test_config: #each iteration is taking a testing case from test_confic
        columns_list = []#to nullify from the previous iteration
        metrics_list = []
        test_config2 = []
        columns_list.append(el[0])
        for k,v in el[1].items():
            metrics_list.append(k)

        test_config2: list = [el] #listing a tuple with 1 case out of many or 1 from

        ml = metrics_list[:]
        expected_df = ff.metadata_info(metrics_list, columns_list) #calculating the
        # here is creting the dataframe with the test_case from test_config
        lfunc = 'lambda x: (x.'+ mock_df_schema[0] +','
        lfunc += ' , '.join(['x.' + mock_df_schema[1] + '["' + ml[i] + '"]' for i in
        lfunc += ')'

        testdf = spark.createDataFrame(data = test_config2, schema = mock_df_schema)
        testdf = testdf.rdd.map(eval(lfunc)).toDF(metrics_list)
        #excdeption handling
        try: #comparing expected datafame and dataframe, made on test case, provided
            assert testdf.toPandas().to_csv() == expected_df.toPandas().to_csv()
            results.append(f"""{str(el[0])} - {str(el[1])} - {'passed'}""")
        except Exception as x:
            results.append(f"""{str(el[0])} - {str(el[1])} - {'failed'}""")
            print("There was an exception with testing " + str(el[0]) + ' ' + str(el
                        "\n" + "ERROR : " + str(x))
    return results
```

In [1445]:

```
#what needs to be mentioned - the file
#with the changes to be done on the dataframe
#testing configuration - as mentioned in test_config - field itself and the values c
#need to be mentioned as per format mentioned below

af = 'nyc-jobs.csv'
afp = '/dataset'
afh = True
afdtcd = datachange

test_config: list =  [('posting_date', {'min_value':'2012-01-26', 'max_value':'2019-
                     ('job_id', {'min_value':87990, 'max_value':426238})]


test_functions(af, afp, afh, afdtcd, test_config)
```

Out[1445]:

```
["posting_date - {'min_value': '2012-01-26', 'max_value': '2019-12-1
7'} - passed",
 "job_id - {'min_value': 87990, 'max_value': 426238} - passed"]
```