Table of Contents

Modeling Benefits Programs for City Employees

Authors: Harini Lakshmanan, Uyen Pham, Stephen Reagin

Company Name: Clever Compensation Company Name [C3N]

Company Industry: HR consulting

Company Size: Three people

Abstract

We are an HR consulting firm which has been tasked by several municipalities in California to perform an audit on the compensation practices for municipal employees. We focus on building regression models to predict the dollar value of employee Benefits.

Problem Statement

Municipal budgets are funded through taxpayer resources, and voters have a right to know how their resources are being allocated for city employment practices. Several cities in California, notably Los Angeles, San Francisco, and San Jose have all released their city payroll 10-year historical information for public consumption. Auditing this information is vital to ensure the public trust in continuing to fund municipal employees for the services they perform on behalf of the public.

As an HR consulting firm, we see this as an opportunity to understand the current state of affairs, recognize the historical trends in pay practices, and identify areas for improvement. The open data sources allow us to look for trends in overall pay practices, as well as take a nuanced dive into differentiation by geographic region and by department.

Goals

The first goal is to review overall pay practices in the select municipalities, including trends and breakouts by department and region. We also want to understand the different components of compensation, e.g. salary, bonus, overtime, benefits, etc. This will allow municipalities to better understand their payroll budgets on a going-forward basis.

In addition, we apply predictive modeling techniques to the monetary value of Total Benefits programs for each city employee using simple regression models. This can be used to assess: (A) the robustness of the dataset, and (B) the extent to which department and cash compensation are likely to predict the benefits that a city employee can expect to receive. If we find very low error in our predictive model, meaning the benefits value is easy to predict, then we can develop tools for educating employees to self-identify possible issues with unfair pay. On the other hand, if we find very high errors, we can further consult with the cities to develop more targeted philosophies regarding overall compensation and fairness practices.

Non-Goals

To limit the scope of our compensation review, for this phase we will not perform anomaly detection, to see whether any groups or individuals have been historically underfunded relative to their municipal counterparts.

We are also intentionally not tying crime statistics to police department budgets. While we recognize this is an active area of interest and research, and that it would be a great next step for further development, such an undertaking would be outside the scope of our company's objective to audit current pay practices.

In addition, we are not predicting or recommending any overall changes to (or reallocations of) municipal budgets, nor are we making recommendations for line-item cuts, because those matters should be decided by a City Council working in partnership with a Mayor's office, all of whom are answerable to voters.

Data Sources

The data for analysis was stored in an AWS S3 bucket at s3://508-team4/, with the data files located in a folder under s3://508-team4/data/ and later ingested into the Amazon Sagemaker platform. The original compensation data for city employees comes from three municipal open data sources, and Consumer Price Index (CPI) comes from the Federal Reserve of Minneapolis:

- Los Angeles, City Employee Payroll, updated quarterly by city Controller https://controllerdata.lacity.org/Payroll/City-Employee-Payroll-Current-/g9h8-fvhu

    - 1 CSV file with 753,458 rows and 18 columns

- San Jose, Open data portal, Employee Compensation Plans

  https://data.sanjoseca.gov/dataset/employee-compensation-plan

    - 10 CSV files (1 per year), each having 7500 - 8500 rows and 12 columns

- San Francisco, DataSF provided by SF Controller Office

  https://data.sfgov.org/City-Management-and-Ethics/Employee-Compensation/88g8-5mnd

    - 1 CSV file with 799,652 rows and 22 columns

- Consumer Price Index 1913 to present

  https://www.minneapolisfed.org/about-us/monetary-policy/inflation-calculator/consumer-price-index-1913-

    - 1 CSV file with 9 rows and 3 columns

Data Exploration

As stated under the Data Sources section, the data files for analysis were stored in an AWS S3 bucket and later ingested into the Amazon Sagemaker platform. We desired to use Amazon Athena as a data warehousing solution and query service; however, due to inconsistent naming conventions, data types, and existing commas, the data had to be preprocessed for Athena use. The team used both Athena and Jupyter Notebook to create a database, perform SQL logic and transformations, and explore the data. Tables from the database were merged to create a single working database of 1,438,685 rows and 15 columns from which further exploration was performed.

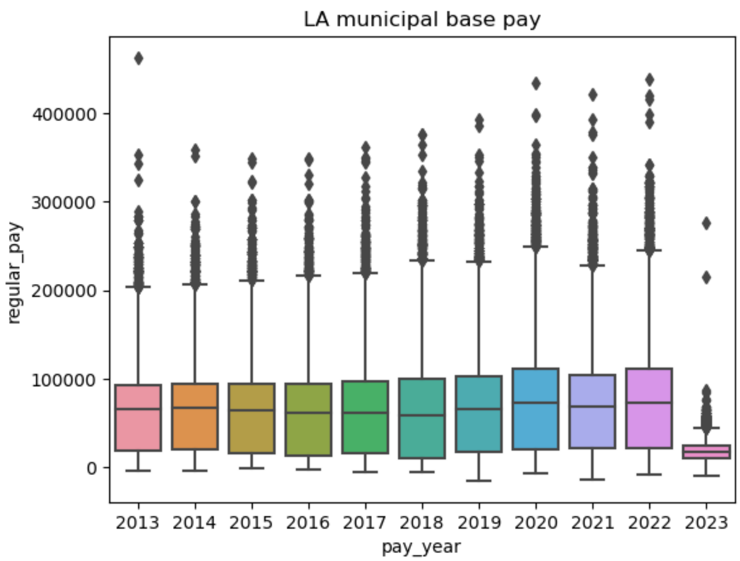
The data quality of the working data frame was carefully assessed, and several steps were taken to address these issues. Firstly, 27,904 duplicated rows were identified and removed from the dataset to ensure data integrity. Additionally, there were missing values in multiple columns that were imputed using relevant methods. For example, department missing values were imputed based on job title, base_salary was imputed using its relationship with total cash and other sources of cash, and overtime and irregular cash were computed with zero values assuming they were not available for certain individuals.

Additionally, 5882 rows contained negative values for either cash or benefits which were inconsistent with the nature of the data. These values could be due to raw input errors and were removed to ensure data accuracy (Hastie et al. 2017).

Furthermore, many outliers were observed in all cities, indicating a wide range of cash and benefits paid to employees. Outliers for Los Angeles could be seen in Figure 1 . These outliers may seem unusual, but they are reliable and important data points that reflect the reality of pay scale differences in various job categories. They provide valuable insights into the compensation practices of different industries and help to identify patterns and trends.

Figure 1

Total regular pay each year in Los Angeles



Data Ingestion

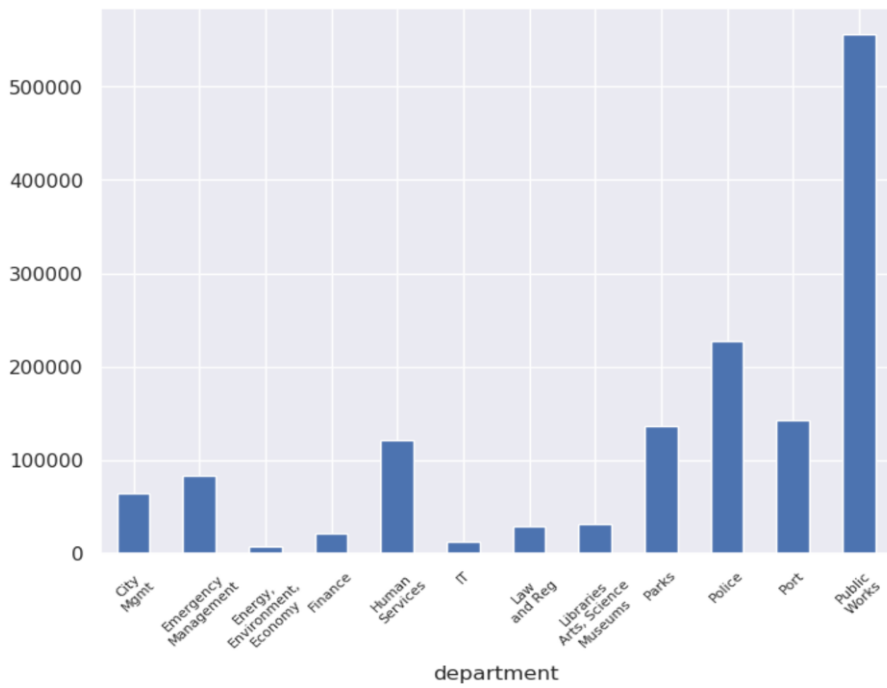
Data ingestion and data exploration were performed using Athena and SageMaker studio notebook. The code is stored in a team GitHub repository located at: https://github.com/unpham/ads-508-project/tree/main and the notebooks discussed in this report are located in the Notebooks subdirectory: https://github.com/unpham/ads-508-project/tree/main/Notebooks

Data Preparation

The Department column contained more than 5,000 unique values, which could have overwhelmed the machine learning process. Therefore, a careful consideration was given to identifying similarities in job nature and grouping departments accordingly. As a result, the departments were condensed into twelve categories, and their distribution can be viewed in Figure 2. Public work seemed to be the dominant category.

Figure 2

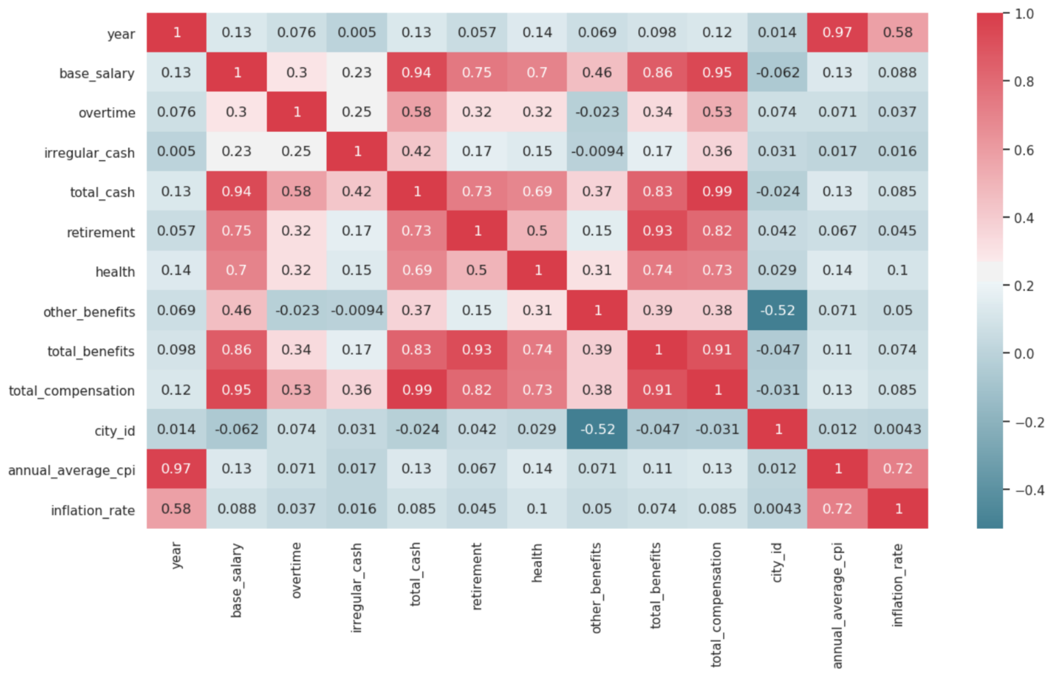
Department distribution



To ensure a more focused and meaningful analysis and achieve our business goals effectively, a combination of methods was used to select the relevant features for the analysis. These methods included using the correlation matrix to identify any features with high correlation (>0.75) that might interfere with our analysis (Figure 3) and selecting features based on their relevance to our business goal.

Figure 3

Feature correlation heatmap



We did not select ethnicity because disclosure and methodology were not uniform across different cities. We did not select employee names because year-over-year changes were inconsistent and difficult to track; for example, changes in surname due to marriage. We created numerical features for “total cash” and “total benefits” as simple aggregations of the cash and benefits components, respectively. The final dataset included:

- Year (category)
- Department (category)

- Base Salary (float)
- Overtime (float)
- Irregular cash (float)
- City (category)
- Annual CPI (float)
- **Total Benefits** (float) as the target value

The descriptive summary statistics for numerical features are shown in Table 1.

Table 1

Summary statistics of the population dataset

	base_salary	overtime	irregular_cash	total_cash	retirement	health	other_benefits	total_benefits	total_compensation
count	1460661.0	1460661.0	1460661.0	1460661.0	1460661.0	1460661.0	1460661.0	1460661.0	1460661.0
mean	68399.0	7209.0	4278.0	79891.0	16654.0	10069.0	2767.0	29489.0	109380.0
std	48137.0	16969.0	9296.0	58959.0	16519.0	7012.0	3742.0	22153.0	78404.0
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
25%	23581.0	0.0	0.0	26760.0	2407.0	2880.0	0.0	7983.0	36316.0
50%	68548.0	56.0	1219.0	76780.0	14247.0	12424.0	418.0	30779.0	108807.0
75%	101285.0	6154.0	4875.0	116735.0	23737.0	15335.0	5300.0	43093.0	161161.0
max	651937.0	434394.0	2394972.0	2394972.0	213678.0	255615.0	35691.0	255615.0	2394972.0

Average Annual Consumer Price Index (CPI) data was used to normalize the cash and benefit values to present year (2021) ensuring that fair comparisons were made across different time periods. This conversion helps adjust for the effects of inflation, resulting in a clearer understanding of the true value of these financial values and enabling more informed decision-making. Average annual CPI values are shown in Figure 4, and comparisons to CPI-adjusted data are shown in Figure 5.

Figure 4

CPI Index from 2013 to 2021

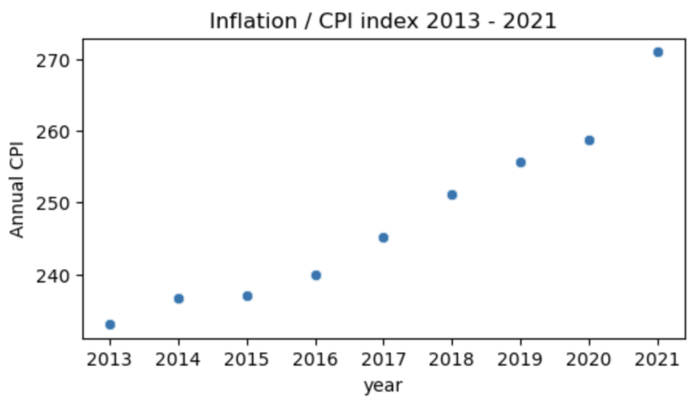
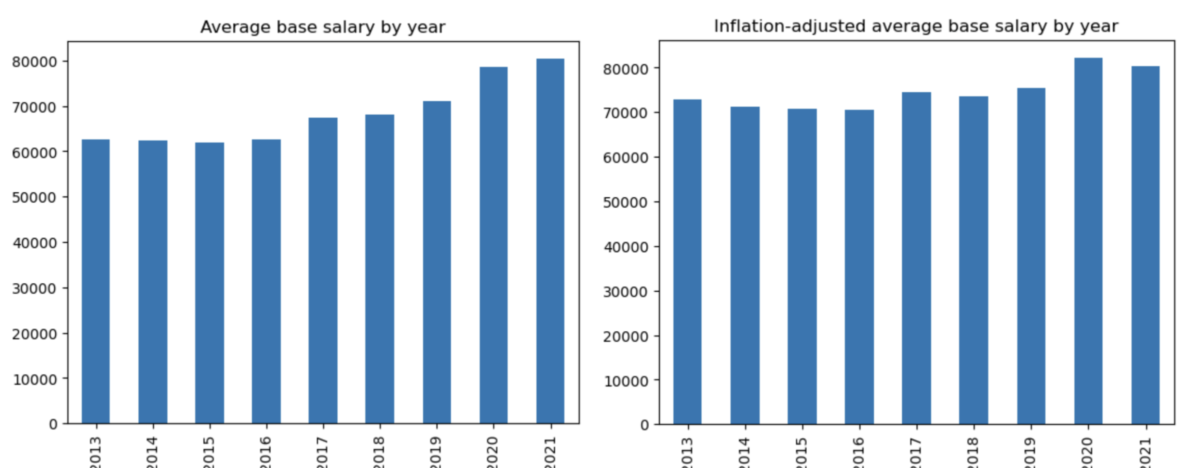


Figure 5

Base Salary and Inflation-Adjusted Salary Over Time

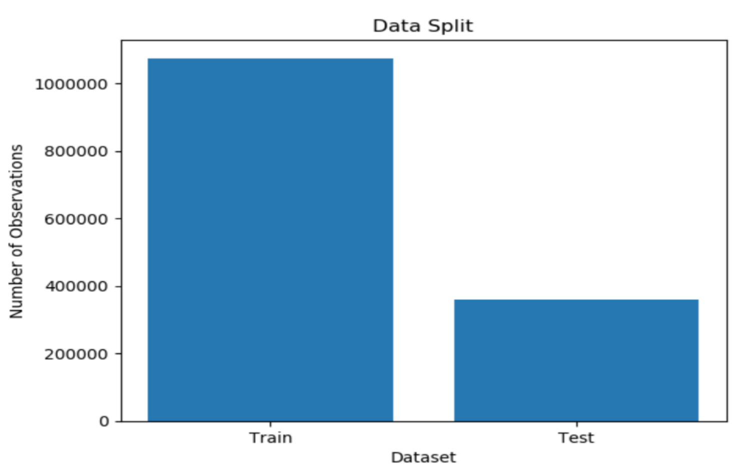


One hot encoding is a technique used to convert categorical features into numerical data, making it easier for machine learning algorithms to process them (Deng, 2014). In addition to one hot encoding, the data was standardized via scaling methods such as the RobustScaler method, which is particularly helpful for data that contain outliers (Sklearn, 2021). Standardization helps with algorithms that are sensitive to the scale of the features, ensuring that each feature is treated equally and preventing any one feature from having too much influence on the overall analysis. By using these methods, the accuracy and performance of machine learning models can be improved, allowing them to make more accurate predictions based on a wider range of features (Bishop, 2006).

The data was split into 75:25 train, test ratio as in Figure 6.

Figure 6

Train and test split



Model Training

We trained several regression models to compare AWS SageMaker algorithms against local Python scripts using the scikit-learn package. Each model had similar initial setups, including the splitting of data into training and testing datasets with the standard labels of X_train, X_test, y_train, y_test. The instance size in all cases was the entire training dataset, representing approximately 75% or 1 million observations of the original 1.4 million observations.

The chosen SageMaker algorithm was the Linear Learner estimator. Following Vegiraju (2020), the set of hyperparameters included an instance type of *ml.c4.xlarge*, a *mini_batch_size* of 20 with 5 epochs and 10 models being trained in parallel with the *regressor* predictor type. The loss function was absolute loss, and the training took approximately 36 minutes to complete.

Normally after training a model, the next step is to create predictions using a test set of data, which in this case refers to the scaled X_test dataset. However, due to a quirk in the AWS ecosystem, attempting to predict more than 20,000 instances at a time resulted in a 413 error, meaning the dataset was too large to be processed. Therefore, we made predictions on the dataset in 20,000-observation increments and stitched the predictions together on the backend.

The local Python scripts were written to run multiple algorithms from the scikit-learn library, including Decision Tree Regression, Lasso Regression, Linear Regression, XGboost, Random Forest Regression, and Ridge Regression. The local scripts did not take advantage of the tools AWS has to offer for optimizing predictive models, but they were still run on AWS servers in SageMaker notebooks.

After training the Linear Learner and local Python models, we also created a regression model using SageMaker Autopilot. This came with benefits such as automatically generating analysis notebooks, storing processing jobs, feature engineering, and model training.

Model Evaluation

We chose root mean squared error (RMSE) as the metric for assessing numerical predictions for the value of Total Benefits an employee might expect to receive. The RMSE for predictions from the SageMaker Linear Learner estimator was $11,592. While this is 48% less than the $22,153 standard deviation of the original population, it does not inspire confidence that this model would serve as an adequate check for employees who would seek to determine whether or not they are unfairly compensated.

The autopilot results recommended XGboost as the best model with an RMSE of $8,370. However, after applying various methods of transformation and standardization, we found that our script for XGboost and Random Forest produced even better results. The RMSE results for our local Python scripts are in Table 2.

Table 2

RMSE for Each Algorithm

Algorithm	Root Mean Squared Error (RMSE)
Decision Tree	$8,313
Lasso	$11,264
Linear	$11,278
XGBoost	$6,416
Random Forest	$6,152
Ridge	$11,251

Measuring Impact

We created metrics for Total Cash Compensation, Total Benefits Compensation, and Total Compensation by aggregating other fields with a sum. These were necessary features because each city pays benefits in a different manner and with different components, and we needed a set of measurements which our model could reference for all cities. In some cases we also normalized the dollar values according to the CPI to create an equal standard over time.

We also greatly reduced the number of Department categories from more than 500 down to only 12. This was necessary to avoid a highly sparse dataset after one-hot encoding.

Security Checklist, Privacy and Other Risks

This process does not use or store any kind of Public Health Information (PHI) data. We do use some Personally Identifiable Information (PII) data, specifically the names of city employees, but this data already comes directly through the disclosure of municipal governments. Our use of this already-disclosed data does not in any way increase the risk of PII being made available. All data has been stored in S3 buckets, specifically s3://508-team4/ and s3://508-team4/data/

User behavior is not tracked in any capacity, and the process does not store or process credit card data or any financial information except for compensation components paid to city employees.

Compensation is inherently biased by geographic location, number of employees under consideration, job functions being performed, and related factors. We must do our best to ensure we analyze trends and changes in the data instead of simply making raw number comparisons. Furthermore, compensation data must always be discussed within an ethical framework because it is deeply tied to the notion of value. Just because a person works incredibly hard, or works many overtime hours, does not necessarily mean the work being performed is intrinsically valuable. At the same time, we must remember that these data points often represent the entire annual earnings of a person, and care must be taken to treat the subject with respect rather than a mere series of numbers for analysis.

Future Enhancements

If we had access to more information from municipalities we could plausibly improve our predictive models by a considerable amount. Specifically we would want to include fields like employee tenure, employee years of experience, career ladders and career levels of municipal employers (e.g. entry-level, early career, mid-career, senior, executive), better itemization of benefits programs, policies on overtime, etc. Using these additional fields, we could better remediate issues of multicollinearity while still maintaining a large number of independent features for our regression models.

We would also like more time to explore data warehousing options. While we were each able to connect to our own Athena database, we never managed the complexity down to where we could all contribute to the same database in the same S3 bucket. Some of the barriers may be related to IAM role permissions within our student accounts, but we would like more time to fully explore our options and have a single centralized data warehouse solution.

Along similar lines, we would like more time to explore the very many features AWS has to offer. While we did some of the SageMaker built-in features and algorithms, the learning curve for using other regression algorithms was steeper than we anticipated when beginning the project. The AWS documentation is very thorough but also biased towards intermediate and advanced users rather than true first-timers. If we had more time to use each “part” of the

AWS ecosystem separately, rather than trying to combine tools for a collective problem in a short window, we may have found better success. For example, we found interesting results using SageMaker Autopilot that could have informed the next round of analysis, including potential “best” algorithm, dimensionality reduction, missing values, et cetera; but we only got to that result during Week 6, and we didn’t have enough time in the framework of our 7-week course to explore further.

References

Bishop, C. M. (2006). *Pattern recognition and machine learning* (Vol. 4). Springer.

Deng, L. (2014). *A tutorial survey of architectures, algorithms, and applications for deep learning.* IEEE Signal Processing Magazine, 31(3), 94-114.

Hastie, T., Tibshirani, R., & Friedman, J. (2017). *The elements of statistical learning: data mining, inference, and prediction*. Springer.

Sklearn. (2021). *Preprocessing data*. Retrieved from: https://scikit-learn.org/stable/modules/preprocessing.html

U.S. Equal Employment Opportunity Commision. (n.d.) *Equal Pay/Compensation Discrimination*. Retrieved from: https://www.eeoc.gov/equal-paycompensation-discrimination

Vegiraju, Ram. (2020). *Using AWS SageMaker's Linear Learner to Solve Regression Problems.* Retrieved from: https://towardsdatascience.com/using-aws-sagemakers-linear-learner-to-solve-regression-problems-36732d802ba6

athena_upham

April 17, 2023

[2]:
```
import numpy as np
import pandas as pd
import sqlite3 as sq
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```
sj_df = pd.read_csv('s3://508-team4/data/sj_compensation/sj_compensation.csv')
sj_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71946 entries, 0 to 71945
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   name                71946 non-null  object
 1   department          71946 non-null  object
 2   job_title           71946 non-null  object
 3   total_cash          71946 non-null  float64
 4   base_salary         71247 non-null  float64
 5   overtime            38998 non-null  float64
 6   health              52959 non-null  float64
 7   retired             7404 non-null   object
 8   year                71946 non-null  int64
 9   city_id             71946 non-null  int64
 10  irregular_cash      71946 non-null  float64
 11  retirement          71946 non-null  float64
 12  other_benefits      71946 non-null  float64
 13  total_benefits      71946 non-null  float64
 14  total_compensation  71946 non-null  float64
dtypes: float64(9), int64(2), object(4)
memory usage: 8.2+ MB
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3553:
DtypeWarning: Columns (7) have mixed types.Specify dtype option on import or set
low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

[4]:
```
sj_df['retired'].unique()
```

[4]: array(['Yes', 'No', nan], dtype=object)

[5]:
```
sj_df[sj_df['city_id'].isnull()]
```

[5]: Empty DataFrame
Columns: [name, department, job_title, total_cash, base_salary, overtime, health, retired, year, city_id, irregular_cash, retirement, other_benefits, total_benefits, total_compensation]
Index: []

[52]:
```
row = sj_df[sj_df['job_title'] == 'BdComm Mbr']
#row
```

[7]:
```
col_names_sj = sj_df.columns.tolist()
col_names_sj
```

[7]:
```
['name',
 'department',
 'job_title',
 'total_cash',
 'base_salary',
 'overtime',
 'health',
 'retired',
 'year',
 'city_id',
 'irregular_cash',
 'retirement',
 'other_benefits',
 'total_benefits',
 'total_compensation']
```

[8]:
```
sf_df = pd.read_csv('s3://508-team4/data/sf_compensation/sf_compensation.csv')
#sf_df['city_id'] = 2
sf_df.head()
```

[8]:
```
   organization_group_code job_family_code job_code year_type  year  \
0                        1            1400     1404    Fiscal  2019
1                        1            9700     9703    Fiscal  2019
2                        1            2900     2918    Fiscal  2019
3                        1            2900     2918    Fiscal  2019
4                        1            2900     2905    Fiscal  2019

                        organization_group department_code      department  \
0  Human Welfare & Neighborhood Development             HSA  Human Services
1  Human Welfare & Neighborhood Development             HSA  Human Services
2  Human Welfare & Neighborhood Development             HSA  Human Services
```

```
3  Human Welfare & Neighborhood Development             HSA  Human Services
4  Human Welfare & Neighborhood Development             HSA  Human Services

   union_code                   union  …  salaries overtime  other_salaries  \
0       790.0  SEIU  Local 1021  Misc  …  60720.01     0.00            0.00
1       535.0  SEIU  Local 1021  Misc  …  91677.00     0.00            0.00
2       535.0  SEIU  Local 1021  Misc  …  89106.03     0.00         1540.00
3       535.0  SEIU  Local 1021  Misc  …  85581.11  3355.94          337.75
4       535.0  SEIU  Local 1021  Misc  …  86457.00     0.00         2090.00

   total_salary  retirement  health_and_dental  other_benefits  \
0      60720.01    13653.20           14733.76         4904.34
1      91677.00    17524.20           14733.76         7411.13
2      90646.03    17327.20           14733.76         7401.92
3      89274.80    16359.16           14151.56         7096.21
4      88547.00    16925.97           14733.76         7257.89

   total_benefits  total_compensation  city_id
0        33291.30            94011.31        2
1        39669.09           131346.09        2
2        39462.88           130108.91        2
3        37606.93           126881.73        2
4        38917.62           127464.62        2

[5 rows x 23 columns]
```

[9]:
```
sf_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 799562 entries, 0 to 799561
Data columns (total 23 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   organization_group_code  799562 non-null  int64
 1   job_family_code         799562 non-null  object
 2   job_code                799562 non-null  object
 3   year_type               799562 non-null  object
 4   year                    799562 non-null  int64
 5   organization_group      799562 non-null  object
 6   department_code         799560 non-null  object
 7   department              799560 non-null  object
 8   union_code              799383 non-null  float64
 9   union                   799383 non-null  object
 10  job_family              799562 non-null  object
 11  job                     799557 non-null  object
 12  employee_identifier     799562 non-null  int64
 13  salaries                799562 non-null  float64
```

```
 14  overtime                  799562 non-null  float64
 15  other_salaries            799562 non-null  float64
 16  total_salary              799562 non-null  float64
 17  retirement                799562 non-null  float64
 18  health_and_dental         799562 non-null  float64
 19  other_benefits            799562 non-null  float64
 20  total_benefits            799562 non-null  float64
 21  total_compensation        799562 non-null  float64
 22  city_id                   799562 non-null  int64
dtypes: float64(10), int64(4), object(9)
memory usage: 140.3+ MB
```

[10]:
```
col_names_sf = sf_df.columns.tolist()
col_names_sf
```

[10]:
```
['organization_group_code',
 'job_family_code',
 'job_code',
 'year_type',
 'year',
 'organization_group',
 'department_code',
 'department',
 'union_code',
 'union',
 'job_family',
 'job',
 'employee_identifier',
 'salaries',
 'overtime',
 'other_salaries',
 'total_salary',
 'retirement',
 'health_and_dental',
 'other_benefits',
 'total_benefits',
 'total_compensation',
 'city_id']
```

[11]:
```
la_df = pd.read_csv('s3://508-team4/data/la_compensation/la_compensation.csv')
#la_df['city_id'] = 3
la_df.info()
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3553:
DtypeWarning: Columns (0,8) have mixed types.Specify dtype option on import or
set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 753959 entries, 0 to 753958
Data columns (total 22 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   record_nbr          753959 non-null  object
 1   year                753959 non-null  int64
 2   department_no       753959 non-null  int64
 3   department          753959 non-null  object
 4   job_class_pgrade    753415 non-null  object
 5   job_title           753415 non-null  object
 6   employment_type     753959 non-null  object
 7   job_status          753959 non-null  object
 8   mou                 753273 non-null  object
 9   mou_title           753166 non-null  object
 10  base_salary         753959 non-null  float64
 11  overtime            753525 non-null  float64
 12  irregular_cash      753525 non-null  float64
 13  total_cash          753959 non-null  float64
 14  retirement          753959 non-null  float64
 15  health              753959 non-null  float64
 16  gender              750098 non-null  object
 17  ethnicity           746703 non-null  object
 18  total_benefits      753959 non-null  float64
 19  total_compensation  753959 non-null  float64
 20  other_benefits      0 non-null       float64
 21  city_id             753959 non-null  int64
dtypes: float64(9), int64(3), object(10)
memory usage: 126.5+ MB
```

[12]:
```
la_df.head()
```

[12]:
```
      record_nbr  year  department_no       department job_class_pgrade  \
0  303030303632  2017             98  WATER AND POWER           3156-5
1    3030303036  2017             98  WATER AND POWER           9105-5
2  303030313232  2017             98  WATER AND POWER           9602-4
3  303030313632  2017             98  WATER AND POWER           5885-5
4  303030323632  2017             98  WATER AND POWER           3841-5

                job_title employment_type job_status mou  \
0               CUSTODIAN       FULL_TIME     ACTIVE   8
1   UTILITY ADMINISTRATOR       FULL_TIME     ACTIVE   M
2  WATER SERVICES MANAGER       FULL_TIME     ACTIVE   M
3            WTR TRTMT OPR       FULL_TIME     ACTIVE   6
4               ELTL MCHC       FULL_TIME     ACTIVE   8

                         mou_title  …  irregular_cash  total_cash  \
```

```
0  OPERATING MAINTENANCE AND SERVICE UNIT  …         2021.84    62532.13
1              MANAGEMENT EMPLOYEES UNIT  …         6170.49   161685.87
2              MANAGEMENT EMPLOYEES UNIT  …        12504.30   258383.42
3        STEAM PLANT AND WATER SUPPLY UNIT  …        12630.52   121949.85
4  OPERATING MAINTENANCE AND SERVICE UNIT  …         1566.75   125196.24

   retirement   health  gender       ethnicity total_benefits  \
0      3678.0  23508.9  FEMALE        HISPANIC        27186.9
1      9186.0  23508.9  FEMALE  ASIAN AMERICAN        32694.9
2     16228.0  23508.9    MALE           BLACK        39736.9
3      6699.0  23508.9    MALE  ASIAN AMERICAN        30207.9
4      6689.0  23508.9    MALE        HISPANIC        30197.9

  total_compensation  other_benefits  city_id
0           89719.03             NaN        3
1          194380.77             NaN        3
2          298120.32             NaN        3
3          152157.75             NaN        3
4          155394.14             NaN        3

[5 rows x 22 columns]
```

[13]:
```
la_df[la_df['base_salary'].isnull()]
```

[13]:
```
Empty DataFrame
Columns: [record_nbr, year, department_no, department, job_class_pgrade,
job_title, employment_type, job_status, mou, mou_title, base_salary, overtime,
irregular_cash, total_cash, retirement, health, gender, ethnicity,
total_benefits, total_compensation, other_benefits, city_id]
Index: []

[0 rows x 22 columns]
```

[14]:
```
row = la_df[la_df['city_id'] == 240894]
row
```

[14]:
```
Empty DataFrame
Columns: [record_nbr, year, department_no, department, job_class_pgrade,
job_title, employment_type, job_status, mou, mou_title, base_salary, overtime,
irregular_cash, total_cash, retirement, health, gender, ethnicity,
total_benefits, total_compensation, other_benefits, city_id]
Index: []

[0 rows x 22 columns]
```

[15]:
```
import boto3
import sagemaker
```

```python
sess = sagemaker.Session()
bucket = sess.default_bucket()
role = sagemaker.get_execution_role()
region = boto3.Session().region_name
```

[16]:
```python
ingest_create_athena_db_passed = False
```

[17]:
```python
sj_df['city_id'].value_counts()
```

[17]:
```
1    71946
Name: city_id, dtype: int64
```

0.1 Import PyAthena

[18]:
```python
!pip install --disable-pip-version-check -q PyAthena==2.1.0
from pyathena import connect
```

```
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is recommended to
use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

0.2 Create Athena Database

[19]:
```python
database_name = "compens"
```

[20]:
```python
# Set S3 staging directory -- this is a temporary directory used for Athena␣
↪queries
s3_staging_dir = "s3://{0}/athena/staging".format(bucket)
```

[21]:
```python
conn = connect(region_name=region, s3_staging_dir=s3_staging_dir)
```

[22]:
```python
statement = "CREATE DATABASE IF NOT EXISTS {}".format(database_name)
print(statement)
```

```
CREATE DATABASE IF NOT EXISTS compens
```

[23]:
```python
pd.read_sql(statement, conn)
```

[23]:
```
Empty DataFrame
Columns: []
Index: []
```

0.3 Verify The Database Has Been Created Succesfully

[24]:
```python
statement = "SHOW DATABASES"

df_show = pd.read_sql(statement, conn)
df_show.head(5)
```

[24]:
```
  database_name
0       compens
1       default
2        dsoaws
```

0.4 Create Tables

0.4.1 San Jose Table

[25]:
```python
#Directory for the data input
data_dir = 's3://508-team4/data'
```

[26]:
```python
table_name1='sj_compensation'
pd.read_sql(f'DROP TABLE IF EXISTS {database_name}.{table_name1}', conn)


create_table = f"""
CREATE EXTERNAL TABLE IF NOT EXISTS {database_name}.{table_name1}(
                name string,
                department string,
                job_title string,
                total_cash float,
                base_salary float,
                overtime float,
                health float,
                retired string,
                year int,
                city_id int,
                irregular_cash float,
                retirement float,
                other_benefits float,
                total_benefits float,
                total_compensation float
 )
                ROW FORMAT DELIMITED
                FIELDS TERMINATED BY ','
                LOCATION '{data_dir}/{table_name1}'
                TBLPROPERTIES ('skip.header.line.count'='1')
"""

pd.read_sql(create_table, conn)
```

```
pd.read_sql(f'SELECT * FROM {database_name}.{table_name1} LIMIT 5', conn)
```

[26]:
```
                 name    department         job_title  total_cash  \
0   Bustillos Steven D        Police   Police Sergeant   286137.70
1       Figone Debra J  City Manager    City Manager U   248564.84
2      Guerra Daniel P        Police    Police Officer   241039.12
3  Moore Christopher M        Police  Chief Of Police U   233540.31
4     Vasquez Hector M        Police    Police Officer   230469.84

   base_salary   overtime    health retired  year  city_id  irregular_cash  \
0     90888.00   89867.88  13640.50     Yes  2013        1       105381.81
1    227975.02        NaN  15166.00     Yes  2013        1        20589.82
2     97198.40  132104.55  15371.56      No  2013        1        11736.18
3     15319.54        NaN   1371.81     Yes  2013        1       218220.78
4     97198.40  124552.69  16345.32      No  2013        1         8718.76

   retirement  other_benefits  total_benefits  total_compensation
0    74429.71             0.0        88070.21            374207.9
1   151550.08             0.0       166716.08            415280.9
2    79821.51             0.0        95193.07            336232.2
3    11204.16             0.0        12575.97            246116.3
4    79821.51             0.0        96166.83            326636.7
```

[27]:
```
# pd.read_sql(f"SELECT * FROM {database_name}.{table_name2} WHERE city_id = 1",␣
↪conn)
```

[ ]:
```
```

0.4.2 San Francisco Table

[28]:
```
table_name2 ='sf_compensation'
pd.read_sql(f'DROP TABLE IF EXISTS {database_name}.{table_name2}', conn)


create_table = f"""
CREATE EXTERNAL TABLE IF NOT EXISTS {database_name}.{table_name2}(
                organization_group_code int,
                job_family_code string,
                job_code string,
                year_type string,
                year int,
                organization_group string,
                department_code string,
                department string,
                union_code float,
```

```
                    union string,
                    job_family string,
                    job_title string,
                    employee_identifier int,
                    base_salary float,
                    overtime float,
                    irregular_cash float,
                    total_cash float,
                    retirement float,
                    health float,
                    other_benefits float,
                    total_benefits float,
                    total_compensation float,
                    city_id int
                    )

                    ROW FORMAT DELIMITED
                    FIELDS TERMINATED BY ','
                    LOCATION '{data_dir}/{table_name2}'
                    TBLPROPERTIES ('skip.header.line.count'='1')
"""

pd.read_sql(create_table, conn)

pd.read_sql(f'SELECT * FROM {database_name}.{table_name2} LIMIT 5', conn)
```

[28]:

```
   organization_group_code job_family_code job_code year_type  year  \
0                        1            2900     2930    Fiscal  2016
1                        1            2900     2930    Fiscal  2015
2                        1            2900     2930    Fiscal  2014
3                        1            2900     2930  Calendar  2016
4                        1            2900     2930  Calendar  2015

  organization_group department_code         department  union_code  \
0   Community Health             DPH  DPH Public Health       790.0
1   Community Health             DPH  DPH Public Health       790.0
2   Community Health             DPH  DPH Public Health       790.0
3   Community Health             DPH  DPH Public Health       790.0
4   Community Health             DPH  DPH Public Health       790.0

                              union  … base_salary overtime  irregular_cash  \
0 SEIU - Miscellaneous  Local 1021  …    83518.12      0.0             0.0
1 SEIU - Miscellaneous  Local 1021  …    76213.74      0.0             0.0
2 SEIU - Miscellaneous  Local 1021  …    17082.00      0.0             0.0
3 SEIU - Miscellaneous  Local 1021  …    86840.71      0.0             0.0
4 SEIU - Miscellaneous  Local 1021  …    79452.81      0.0             0.0
```

```
   total_cash  retirement    health  other_benefits  total_benefits  \
0    83518.12    15542.80  13067.98         6746.82        35357.60
1    76213.74    12503.29  12399.98         6131.47        31034.74
2    17082.00        0.00   2876.46         1347.70         4224.16
3    86840.71    16002.70  13371.04         7015.52        36389.26
4    79452.81    16355.38  12424.50         6412.64        35192.52

   total_compensation  city_id
0           118875.72        2
1           107248.48        2
2            21306.16        2
3           123229.97        2
4           114645.33        2

[5 rows x 23 columns]
```

0.4.3 Los Angeles Table

[29]:
```
table_name3 ='la_compensation'
pd.read_sql(f'DROP TABLE IF EXISTS {database_name}.{table_name3}', conn)


create_table = f"""
CREATE EXTERNAL TABLE IF NOT EXISTS {database_name}.{table_name3}(
                record_nbr string,
                year int,
                department_no string,
                department string,
                job_class_pgrade int,
                job_title string,
                employment_type string,
                job_status string,
                mou string,
                mou_title string,
                base_salary float,
                overtime float,
                irregular_cash float,
                total_cash float,
                retirement float,
                health float,
                gender string,
                ethnicity string,
                total_benefits float,
                total_compensation float,
                other_benefits float,
                city_id int
                )
```

```
                ROW FORMAT DELIMITED
                FIELDS TERMINATED BY ','
                LOCATION '{data_dir}/{table_name3}'
                TBLPROPERTIES ('skip.header.line.count'='1')
"""

pd.read_sql(create_table, conn)

pd.read_sql(f'SELECT * FROM {database_name}.{table_name3} LIMIT 5', conn)
```

[29]:

```
      record_nbr  year department_no            department job_class_pgrade  \
0  303536333832  2015            88  RECREATION AND PARKS             None
1  303536333932  2015            70                POLICE             None
2    3035363434  2015            70                POLICE             None
3  303536343733  2015             4              AIRPORTS             None
4    3035363437  2015            42                HARBOR             None

                     job_title employment_type  job_status mou  \
0         RECREATION ASSISTANT       PART_TIME  NOT_ACTIVE  07
1            POLICE OFFICER II       FULL_TIME  NOT_ACTIVE  24
2  SENIOR MANAGEMENT ANALYST I       FULL_TIME  NOT_ACTIVE  20
3           VOCATIONAL WORKER I       FULL_TIME  NOT_ACTIVE  04
4         PORT POLICE SERGEANT       FULL_TIME      ACTIVE  38

                               mou_title  …  irregular_cash  total_cash  \
0                  RECREATION ASSISTANTS  …           10.40     1711.27
1  POLICE OFFICERS  LIEUTENANT AND BELOW  …         4166.60    99018.29
2            SUPERVISORY ADMINISTRATIVE  …         2535.74   126754.58
3          EQUIPMENT OPERATION AND LABOR  …          309.10     7127.56
4         LOS ANGELES PORT POLICE ASSOC.  …        16243.87   173021.60

   retirement    health  gender  ethnicity total_benefits total_compensation  \
0        0.00      0.00    MALE   HISPANIC           0.00            1711.27
1    42506.82  10330.97  FEMALE  CAUCASIAN       52837.79          151856.08
2    32472.95   6393.84  FEMALE      BLACK       38866.79          165621.38
3     1758.57   1104.76  FEMALE   HISPANIC        2863.33            9990.89
4    61461.57  16641.84    MALE  CAUCASIAN       78103.41          251125.02

   other_benefits  city_id
0             NaN        3
1             NaN        3
2             NaN        3
3             NaN        3
4             NaN        3
```

[5 rows x 22 columns]

0.4.4 cpi table

[39]:
```
table_name4 ='cpi'
pd.read_sql(f'DROP TABLE IF EXISTS {database_name}.{table_name4}', conn)


create_table = f"""
CREATE EXTERNAL TABLE IF NOT EXISTS {database_name}.{table_name4}(\
                year int, \
                annual_average_cpi float, \
                inflation_rate float)

                ROW FORMAT DELIMITED
                FIELDS TERMINATED BY ','
                LOCATION '{data_dir}/{table_name4}'
                TBLPROPERTIES ('skip.header.line.count'='1')
"""

pd.read_sql(create_table, conn)

pd.read_sql(f'SELECT * FROM {database_name}.{table_name4} LIMIT 5', conn)
```

[39]:

	year	annual_average_cpi	inflation_rate
0	2013	233.0	1.5
1	2014	236.7	1.6
2	2015	237.0	0.1
3	2016	240.0	1.3
4	2017	245.1	2.1

[40]:
```
#Check tables in the database
df_show = pd.read_sql('SHOW TABLES IN compens', conn)
df_show.head(5)
```

[40]:

	tab_name
0	cpi
1	la_compensation
2	sf_compensation
3	sj_compensation

0.5 Merge the compensation and cpi tables

[41]:
```
# The 3 table were merge based on the columns of interest and join to cpi table␣
↪based on year column
df2 = pd.read_sql(f'SELECT t.*, t4.annual_average_cpi, t4.inflation_rate \
        FROM (SELECT year, department, job_title, base_salary, overtime,␣
↪irregular_cash, \
            total_cash, retirement, health, other_benefits, total_benefits,␣
↪total_compensation, city_id \
        FROM {database_name}.{table_name1} \
        UNION ALL SELECT year, department, job_title, base_salary, overtime,␣
↪irregular_cash, \
            total_cash, retirement, health, other_benefits, total_benefits,␣
↪total_compensation, city_id \
        FROM  {database_name}.{table_name2} \
        UNION ALL SELECT year, department, job_title, base_salary, overtime,␣
↪irregular_cash, \
            total_cash, retirement, health, other_benefits, total_benefits,␣
↪total_compensation, city_id \
        FROM  {database_name}.{table_name3}) t \
        JOIN {database_name}.{table_name4} t4 ON t.year = t4.year', conn)
```

[42]:
```
df2.tail()
```

[42]:
```
         year                                   department  \
1466584  2019  ECONOMIC AND WORKFORCE DEVELOPMENT DEPARTMENT
1466585  2019                                     AIRPORTS
1466586  2019                                     AIRPORTS
1466587  2019                          RECREATION AND PARKS
1466588  2019                                       POLICE

                      job_title  base_salary  overtime  irregular_cash  \
1466584  SENIOR PROJECT ASSISTANT     21177.60      0.00            0.00
1466585      MANAGEMENT ASSISTANT     42070.40   1074.76          500.00
1466586          SECURITY OFFICER     20813.04    732.79          805.00
1466587         GARDENER CARETAKER     58024.00   1944.42         1239.27
1466588          POLICE OFFICER II    107443.20  10400.97         1525.00

         total_cash  retirement    health  other_benefits  total_benefits  \
1466584    21177.60     6281.28   1140.28             NaN         7421.56
1466585    43645.16    12478.08   8155.86             NaN        20633.94
1466586    22350.83     6173.15   4419.44             NaN        10592.59
1466587    61207.69    17209.92  19731.84             NaN        36941.76
1466588   119369.17    50337.14   9831.64             NaN        60168.78

         total_compensation  city_id  annual_average_cpi  inflation_rate
1466584            28599.16        3               255.7             1.8
```

```
1466585      64279.10    3    255.7    1.8
1466586      32943.42    3    255.7    1.8
1466587      98149.45    3    255.7    1.8
1466588     179537.95    3    255.7    1.8
```

[43]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1466589 entries, 0 to 1466588
Data columns (total 15 columns):
 #   Column               Non-Null Count    Dtype
---  ------               --------------    -----
 0   year                 1466589 non-null  int64
 1   department           1466589 non-null  object
 2   job_title            1466589 non-null  object
 3   base_salary          1465890 non-null  float64
 4   overtime             1433207 non-null  float64
 5   irregular_cash       1466155 non-null  float64
 6   total_cash           1466589 non-null  float64
 7   retirement           1466589 non-null  float64
 8   health               1447602 non-null  float64
 9   other_benefits       830557 non-null   float64
 10  total_benefits       1466589 non-null  float64
 11  total_compensation   1466589 non-null  float64
 12  city_id              1466589 non-null  int64
 13  annual_average_cpi   1466589 non-null  float64
 14  inflation_rate       1466589 non-null  float64
dtypes: float64(11), int64(2), object(2)
memory usage: 167.8+ MB
```

[50]:
```
#df.to_csv('compensation_cpi.csv.gz', index=False, compression='gzip')
```

[45]:
```
conn.close()
```

[53]:
```
%%html

<p><b>Shutting down your kernel for this notebook to release resources.</b></p>
<button class="sm-command-button" data-commandlinker-command="kernelmenu:
↪shutdown" style="display:none;">Shutdown Kernel</button>

<script>
try {
    els = document.getElementsByClassName("sm-command-button");
    els[0].click();
}
catch(err) {
    // NoOp
```

```
}
</script>
```

[ ]:

```
[ ]:
```

[54]:

```
%%javascript

try {
    Jupyter.notebook.save_checkpoint();
    Jupyter.notebook.session.delete();
}
catch(err) {
    // NoOp
}
```

[ ]:

```
```

eda_models_sfr_uph

April 17, 2023

[48]:
```
import numpy as np
import pandas as pd
import sqlite3 as sq
import matplotlib.pyplot as plt
import seaborn as sns
import gzip
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.linear_model import LinearRegression,  Ridge, Lasso
from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
 ↪mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.ensemble import RandomForestRegressor
import random
import warnings
warnings.filterwarnings('ignore')
```

[49]:
```
with gzip.open('compensation_cpi.csv.gz', 'rb') as f:
    df2= pd.read_csv(f)
#df2.head()
```

[50]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1466589 entries, 0 to 1466588
Data columns (total 15 columns):
 #   Column              Non-Null Count    Dtype
---  ------              --------------    -----
 0   year                1466589 non-null  int64
 1   department          1466587 non-null  object
 2   job_title           1466043 non-null  object
 3   base_salary         1465890 non-null  float64
 4   overtime            1433207 non-null  float64
 5   irregular_cash      1466155 non-null  float64
 6   total_cash          1466589 non-null  float64
 7   retirement          1466589 non-null  float64
 8   health              1447602 non-null  float64
```

```
 9   other_benefits      830557 non-null   float64
 10  total_benefits      1466589 non-null  float64
 11  total_compensation  1466589 non-null  float64
 12  city_id             1466589 non-null  int64
 13  annual_average_cpi  1466589 non-null  float64
 14  inflation_rate      1466589 non-null  float64
dtypes: float64(11), int64(2), object(2)
memory usage: 167.8+ MB
```

0.1 Check for duplicates

[51]:
```python
#Count the number of duplicates
num_duplicates = df2.duplicated().sum()
num_duplicates
# #view duplicate rows
# duplicate_row = df[df.duplicated()]
# duplicate_row.head()
```

[51]:
```
27904
```

[52]:
```python
# drop duplicates and reset index
df2 = df2.drop_duplicates().reset_index(drop=True)
df2.shape
```

[52]:
```
(1438685, 15)
```

0.2 Check for missing data

[53]:
```python
# Check missing data
df2.isnull().sum()
```

[53]:
```
year                       0
department                 2
job_title                446
base_salary              680
overtime               33113
irregular_cash           406
total_cash                 0
retirement                 0
health                 18833
other_benefits        618294
total_benefits             0
total_compensation         0
city_id                    0
annual_average_cpi         0
inflation_rate             0
dtype: int64
```

–> The null values in compensation columns could be due to the fact that those values are not applicable such as people did not have overtime work, or contractors that might not subject to compensation. It is safe to impute those missing value with 0

[54]:
```
#Impute missing value in salary an dcompensation with 0 value
df2[['overtime', 'irregular_cash' ,  'health', 'other_benefits']] = df2[[
 ↪'overtime', 'irregular_cash' ,  'health', 'other_benefits']].fillna(0)
df2['job_title'] = df2['job_title'].fillna("Not disclosed")
df2['base_salary'] = df2['base_salary'].fillna((df2['total_cash'] -
 ↪df2['overtime'] - df2['irregular_cash']))
```

[55]:
```
#observe row with null values in department
dep_null = df2[df2['department'].isnull()]
dep_null.head()
```

[55]:
```
        year department         job_title  base_salary  overtime  \
138816  2017        NaN   Sheriff's Cadet     49630.50  15016.51
425621  2017        NaN  Police Officer 2    116189.62  40990.09

        irregular_cash  total_cash  retirement    health  other_benefits  \
138816         3197.52    67844.53    10619.27  12779.88         4796.56
425621         2260.08   159439.80    20076.66  14515.01         2724.05

        total_benefits  total_compensation  city_id  annual_average_cpi  \
138816        28195.71            96040.24        2               245.1
425621        37315.72           196755.52        2               245.1

        inflation_rate
138816             2.1
425621             2.1
```

[56]:
```
#fill in the missing police department names
df2['department'].fillna('Police',inplace=True)
```

[57]:
```
df2.isna().sum()
df2.isnull().sum()
```

[57]:
```
year                  0
department            0
job_title             0
base_salary           0
overtime              0
irregular_cash        0
total_cash            0
retirement            0
health                0
other_benefits        0
```

```
total_benefits        0
total_compensation    0
city_id               0
annual_average_cpi    0
inflation_rate        0
dtype: int64
```

[58]:
```
#Observe job title
df2['job_title'].nunique()
```

[58]:
```
4175
```

[59]:
```
unique_values2 = df2.groupby('city_id')['job_title'].nunique()
unique_values2
```

[59]:
```
city_id
1     682
2    1360
3    2159
Name: job_title, dtype: int64
```

[60]:
```
# unique_values = df.groupby('department')['job_title'].unique()

# # Print the unique values for each group
# for group, values in unique_values.items():
#     print(f"Group '{group}' has the following unique values in the␣
↪'column_name' column:")
#     print(values)
#     print()
```

–> there are almost 5000 job_titles which would be a challenge as a feature in modeling. In addition condensing is not an easy task, so we might have to drop it, therefore imputing missing data is not neccesary here.

0.3 Condensing Department Names

[61]:
```
def replace_text(text):
    if pd.isna(text) or text is None:
        return text
    elif target_word.lower() in text.lower():
        return new_word
    else:
        return text
```

[62]:
```
target_word= "Police"
new_word= "Police"
```

```
df2['department'] = df2['department'].apply(replace_text)
```

[63]:
```
dept_dict = {
    'Police': 'Police', 'Sheriff': 'Police', "Vcet" : "Police",
    "Fire" : "Emergency Management",    "Emergency" : "Emergency Management",
    "PW" : "Public Works",    "Public" : "Public Works",    "Water" : "Public␣
↪Works",    "DOT" : "Public Works",    "Transport" : "Public Works",
    "Plan" : "Public Works",    "Building" : "Public Works",    #"District" :␣
↪"Public Works",
    "PRNS" : "Parks",    "Recre" : "Parks",    "Zoo" : "Parks",    "Parks" :␣
↪"Parks",    "Arena" : "Parks",
    "City" : "City Mgmt",    "Convention" : "City Mgmt",    "Neighbor" : "City␣
↪Mgmt",    "Election" : "City Mgmt",    "Council" : "City Mgmt",
    "CII" : "City Mgmt",    "Clerk" : "City Mgmt",    "Registrar" : "City␣
↪Mgmt",    "Housing" : "City Mgmt",    "Mayor" : "City Mgmt",    "rda" :␣
↪"City Mgmt",
    "Airport" : "Airport",    "Airside" : "Airport",
    "Finance" : "Finance",    "Auditor" : "Finance",    "Assessor" : "Finance",␣
↪   "Controller" : "Finance",    "Tax" : "Finance", "Treasure" : "Finance",
    "Board" : "Law and Reg",    "Attorney" : "Law and Reg",    "Court" : "Law␣
↪and Reg",
    "Ethics" : "Law and Reg",    "Probation" : "Law and Reg",    "Regulation" :␣
↪"Law and Reg",
    "prt" : "Port",    "port" : "Port", "Harbor" : "Port",
    "Human" : "Human Services",    "Retire" : "Human Services",    "Child" :␣
↪"Human Services",    "Service" : "Human Services",
    "Personnel" : "Human Services",    "Aging" : "Human Services",    "Women" :␣
↪"Human Services",    "Pension" : "Human Services",
    "Disability" : "Human Services",    "Families" : "Human Services", "Youth" :
↪ "Human Services",
    "ESD" : "Human Services",    "Employee" : "Human Services",
    "Info" : "IT",    "Tech" : "IT",
    "Envi" : "Energy, Env, Economy",    "Energy" : "Energy, Env, Economy",   ␣
↪"Power" : "Energy, Env, Economy", "Econ" : "Energy, Env, Economy",
    "Science" : "Libraries, Arts, Science, Museums",    "Librar" : "Libraries,␣
↪Arts, Science, Museums",    "Museum" : "Libraries, Arts, Science, Museums",
    "Memorial" : "Libraries, Arts, Science, Museums",    "Monument" :␣
↪"Libraries, Arts, Science, Museums",    "Arts" : "Libraries, Arts, Science,␣
↪Museums",
    "Cultur" : "Libraries, Arts, Science, Museums", "Art Commission" :␣
↪"Libraries, Arts, Science, Museums"
}
```

[64]:
```
for key in dept_dict:
    target_word= key
    new_word= dept_dict[key]
```

```
    df2['department'] = df2['department'].apply(replace_text)
```

[65]:
```
df2['department'].unique()
```

[65]:
```
array(['Parks', 'City Mgmt', 'Public Works', 'Finance', 'Law and Reg',
       'Port', 'Libraries, Arts, Science, Museums', 'Human Services',
       'Police', 'IT', 'Emergency Management', 'Energy, Env, Economy'],
      dtype=object)
```

0.4 Summary Statistics and outliers

[66]:
```
#Summary sttistics
df2.describe()
```

[66]:
```
               year   base_salary      overtime  irregular_cash    total_cash  \
count  1.438685e+06  1.438685e+06  1.438685e+06    1.438685e+06  1.438685e+06
mean   2.017060e+03  6.939467e+04  7.340716e+03    4.326823e+03  8.106693e+04
std    2.539337e+00  4.780912e+04  1.709815e+04    9.314511e+03  5.864736e+04
min    2.013000e+03 -6.877178e+04 -2.490362e+04   -6.908210e+04 -6.877178e+04
25%    2.015000e+03  2.607120e+04  0.000000e+00    2.470000e+00  2.979432e+04
50%    2.017000e+03  6.940488e+04  9.560000e+01    1.285220e+03  7.790465e+04
75%    2.019000e+03  1.019360e+05  6.421260e+03    4.961200e+03  1.175153e+05
max    2.021000e+03  6.519367e+05  4.343939e+05    2.394972e+06  2.394972e+06

         retirement        health  other_benefits  total_benefits  \
count  1.438685e+06  1.438685e+06    1.438685e+06    1.438685e+06
mean   1.691722e+04  1.020891e+04    2.804429e+03    2.993055e+04
std    1.653596e+04  6.967392e+03    3.754897e+03    2.205499e+04
min   -5.869240e+04 -1.259245e+04   -1.063650e+04   -5.039960e+04
25%    3.089360e+03  3.326620e+03    0.000000e+00    9.083400e+03
50%    1.450748e+04  1.243878e+04    5.029100e+02    3.114874e+04
75%    2.397671e+04  1.539537e+04    5.365810e+03    4.334440e+04
max    2.136775e+05  2.556148e+05    3.569104e+04    2.556148e+05

       total_compensation       city_id  annual_average_cpi  inflation_rate
count        1.438685e+06  1.438685e+06        1.438685e+06    1.438685e+06
mean         1.109975e+05  2.379948e+00        2.477668e+02    1.847782e+00
std          7.793299e+04  5.789799e-01        1.163875e+01    1.158377e+00
min         -7.408261e+04  1.000000e+00        2.330000e+02    1.000000e-01
25%          4.074871e+04  2.000000e+00        2.370000e+02    1.300000e+00
50%          1.102013e+05  2.000000e+00        2.451000e+02    1.600000e+00
75%          1.621956e+05  3.000000e+00        2.557000e+02    2.100000e+00
max          2.394972e+06  3.000000e+00        2.710000e+02    4.700000e+00
```

[67]:

```
#Check rows with negative values
cols = ['base_salary', 'overtime', 'irregular_cash', 'total_cash', ␣
↪'retirement', 'health',        'other_benefits',␣
↪'total_benefits',        'total_compensation', 'city_id']
negative_rows = df2[cols][(df2[cols] < 0).any(axis=1)]
negative_rows.head()
```

[67]:
```
      base_salary  overtime  irregular_cash  total_cash  retirement    health  \
2985    144646.56      0.00        12521.65   157168.20    41550.49  17936.85
4620     21792.19   -292.80           94.68    21594.07        0.00      0.00
8231    119694.97      0.00        11084.23   130779.20    27239.17  15551.14
9500     37013.00    193.93         -599.05    36607.88    10978.06    289.90
9788    163147.23  31827.60        -2665.12   192309.72    76434.48  18511.00

      other_benefits  total_benefits  total_compensation  city_id
2985         -239.02        59248.32           216416.53        2
4620         1676.05         1676.05            23270.12        2
8231         -189.99        42600.32           173379.52        2
9500            0.00        11267.96            47875.84        3
9788            0.00        94945.48           287255.20        3
```

[68]:
```
negative_rows.shape
```

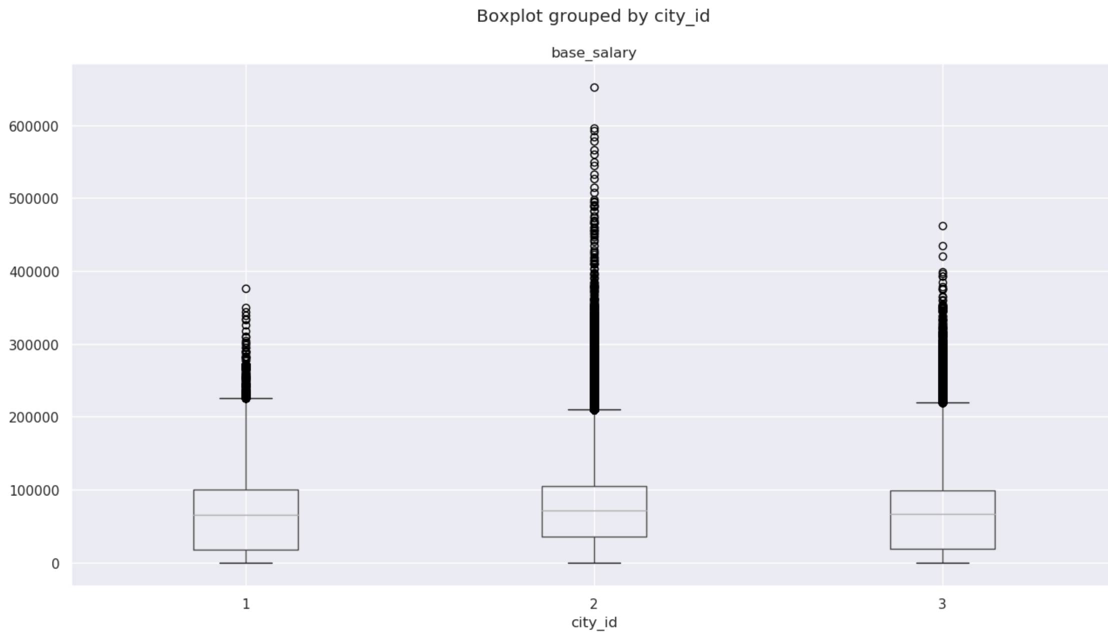
[68]: (5882, 10)

–> there are negative values distributed randomly in most of the float type columns such as base_salary, overtime, etc. It doesn't seem to make sense why would someone working would get negative cash and negative benefit pay. There are roughly 6000 instances which is very small portion of the entire data, so we decided to drop these rows

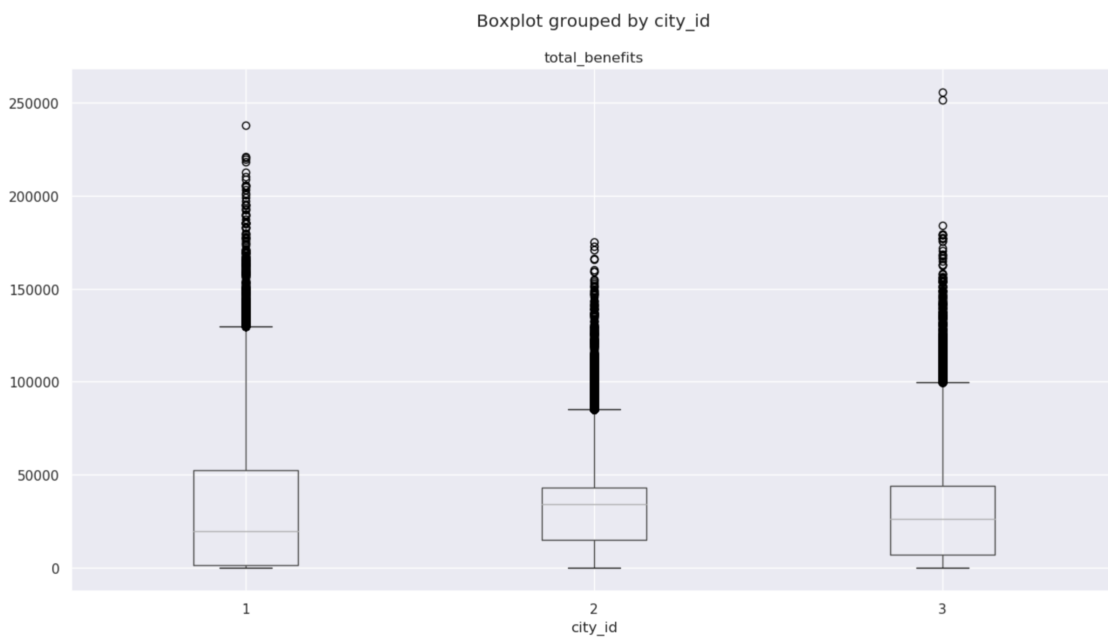
[69]:
```
#Drop those row with negative values
df2 = df2.drop(negative_rows.index).reset_index(drop=True)
df2.shape
```

[69]: (1432803, 15)

[70]:
```
boxplot = df2.boxplot(column=['base_salary'], by='city_id')
plt.show()
```

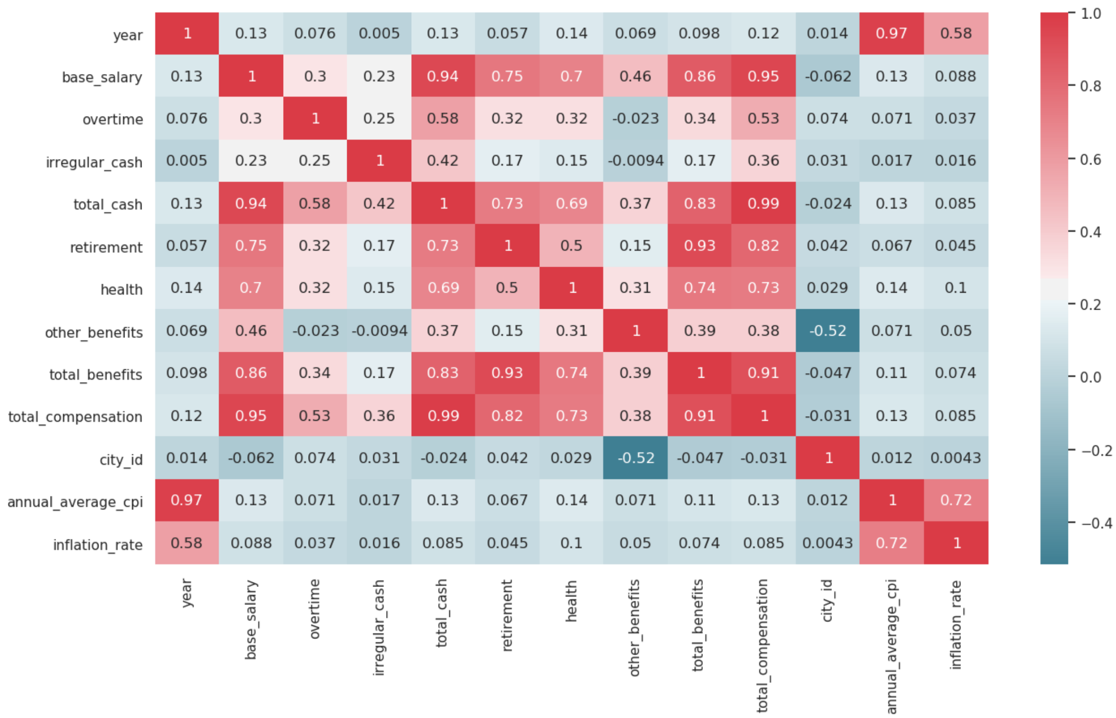


[71]:
```
boxplot = df2.boxplot(column=['total_benefits'], by='city_id')
plt.show()
```



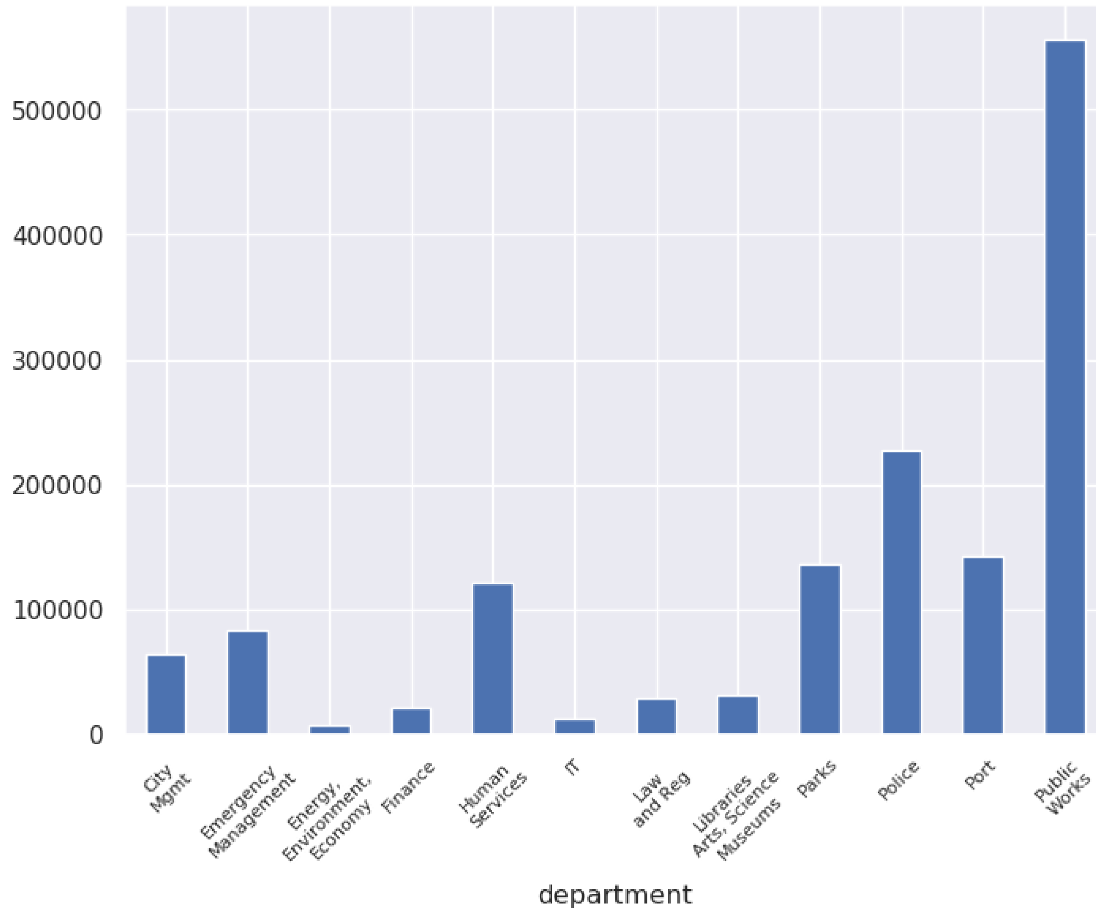
0.5 Examine Correlation and Feature Selection

[72]:
```
# plot the heatmap and annotation on it
#Correlation matrix
corr_matrix = df2.corr()
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.set(rc = {'figure.figsize':(15,8)})
sns.heatmap(corr_matrix, cmap=cmap, annot=True)
plt.show()
```

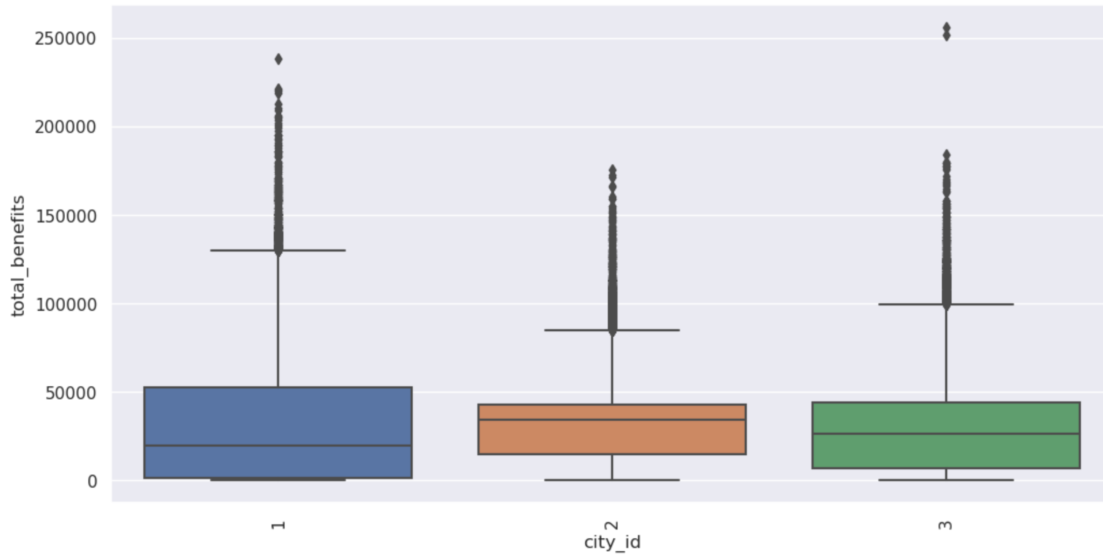


[73]:
```
# group by 'category' and count the values in each group
counts = df2.groupby('department')['department'].count()

#Plot department distribution
fig, ax = plt.subplots(figsize=(8, 6))
counts.plot(kind='bar', ax=ax)
# break the tick labels into two lines
labels = ['City\nMgmt', 'Emergency\nManagement',␣
 ↪'Energy,\nEnvironment,\nEconomy',
          'Finance', 'Human\nServices', 'IT', 'Law\nand Reg',
          'Libraries\nArts, Science\nMuseums', 'Parks', 'Police', 'Port',␣
 ↪'Public\nWorks']
ax.set_xticklabels(labels, fontsize=8)
plt.xticks(rotation=45)
```

```
plt.show()
```



[74]:
```
plt.figure(figsize=(12,6))
sns.boxplot(data=df2,x='city_id',y='total_benefits')
plt.xticks(rotation=90)
plt.show()
```



[75]:
```
#Examine columns with high correlation
cor_matrix = df2.corr().abs()
upper_tri = cor_matrix.where(np.triu(np.ones(cor_matrix.shape),
                                     k=1).astype(bool))

high_corr = [column for column in upper_tri.columns if
             any(upper_tri[column] > 0.75)]
print('These are the columns prescribed to be potentially dropped:␣
 ↪%s'%high_corr)
```

```
These are the columns prescribed to be potentially dropped: ['total_cash',
'retirement', 'total_benefits', 'total_compensation', 'annual_average_cpi']
```

–> since total_benefits would be our target, so it would be kept. Annual_average_cpi is highly correlated with year, but Annual_average_cpi might be a better feature to reflect the anual economy which could affect the salary and benefit.

[76]:
```
#Drop columns
to_drop = ['job_title',  'total_cash',  'retirement',  'health',␣
 ↪'other_benefits', 'inflation_rate',  'total_compensation']
df3 = df2.drop(to_drop, axis=1)
df3.head()
```

[76]:

	year	department	base_salary	overtime	irregular_cash	total_benefits	\
0	2020	Parks	5257.50	0.00	139.32	418.88	
1	2020	City Mgmt	7699.19	1916.90	0.00	746.36	
2	2020	City Mgmt	2619.15	930.50	0.00	275.51	
3	2020	City Mgmt	1870.62	591.22	0.00	191.07	
4	2020	Public Works	158812.14	0.00	5676.94	60283.48	

	city_id	annual_average_cpi
0	2	258.8
1	2	258.8
2	2	258.8
3	2	258.8
4	2	258.8

[77]:
```
#write the data
#df3.to_csv('df_final.csv.gz', index=False, compression='gzip')
```

0.5.1 –> *New data frame df_final.csv.gz*

[83]:
```
with gzip.open('df_final.csv.gz', 'rb') as f:
    df4= pd.read_csv(f)
#df4.head()
```

0.6 Feature engineering

[84]:
```
#convert ctity_id , year to string
df4['city_id'] = df4['city_id'].astype(str)
df4['year'] = df4['year'].astype(str)
# Convert categorical data to dummy values
df4= pd.get_dummies(df4, drop_first=True) # drop first dummy column to avoid␣
↪dummy variable trap
df4.head()
```

[84]:

	base_salary	overtime	irregular_cash	total_benefits	annual_average_cpi	\
0	5257.50	0.00	139.32	418.88	258.8	
1	7699.19	1916.90	0.00	746.36	258.8	
2	2619.15	930.50	0.00	275.51	258.8	
3	1870.62	591.22	0.00	191.07	258.8	
4	158812.14	0.00	5676.94	60283.48	258.8	

	year_2014	year_2015	year_2016	year_2017	year_2018	…	\
0	0	0	0	0	0	…	
1	0	0	0	0	0	…	
2	0	0	0	0	0	…	
3	0	0	0	0	0	…	
4	0	0	0	0	0	…	

	department_Human Services	department_IT	department_Law and Reg	\
0	0	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	

```
4                          0              0                        0

   department_Libraries, Arts, Science, Museums  department_Parks  \
0                                             0                 1
1                                             0                 0
2                                             0                 0
3                                             0                 0
4                                             0                 0

   department_Police  department_Port  department_Public Works  city_id_2  \
0                  0                0                        0          1
1                  0                0                        0          1
2                  0                0                        0          1
3                  0                0                        0          1
4                  0                0                        1          1

   city_id_3
0          0
1          0
2          0
3          0
4          0

[5 rows x 26 columns]
```

[85]:

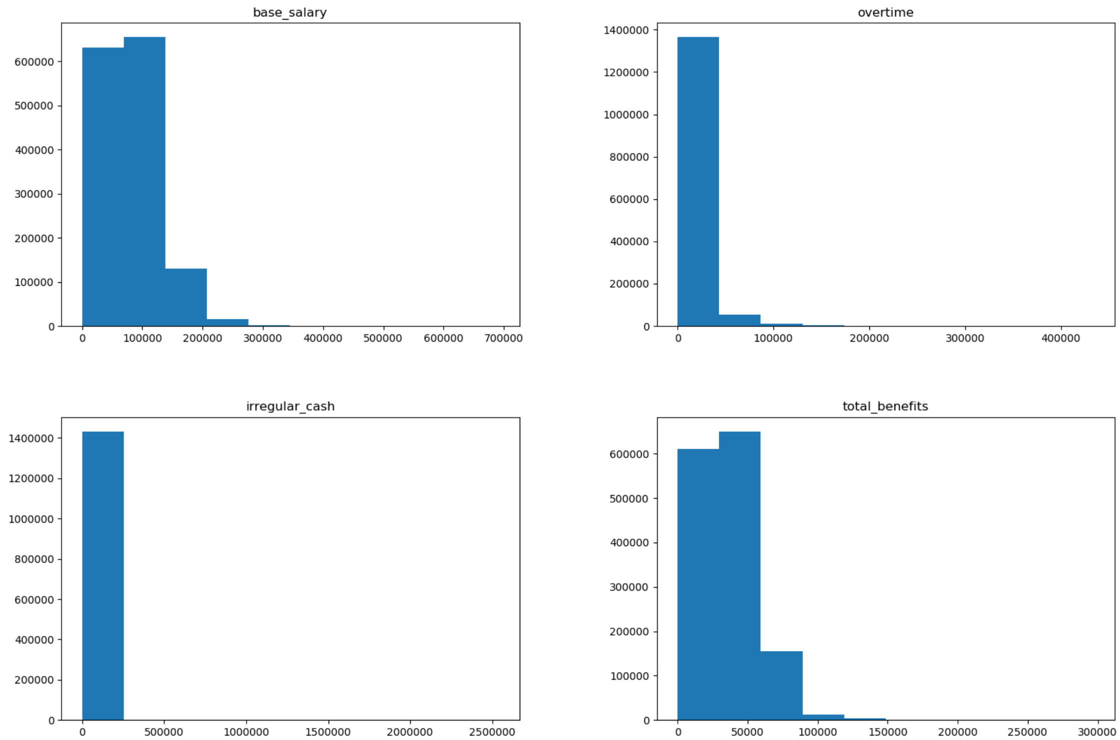
```
#Adjusting all numbers to present-day CPI which is 2021
max_cpi = df4['annual_average_cpi'].max()

for column in ['base_salary', 'overtime', 'irregular_cash',  'total_benefits']:
    df4[column] = df4[column] * (max_cpi / df4['annual_average_cpi'])
df4.head()
```

[85]:

```
     base_salary     overtime  irregular_cash  total_benefits  \
0    5505.341963     0.000000      145.887635      438.626275
1    8062.134815  2007.263910        0.000000      781.543895
2    2742.618431   974.364374        0.000000      288.497720
3    1958.802241   619.090495        0.000000      200.077164
4  166298.647372     0.000000     5944.554637    63125.282380

   annual_average_cpi  year_2014  year_2015  year_2016  year_2017  year_2018  \
0               258.8          0          0          0          0          0
1               258.8          0          0          0          0          0
2               258.8          0          0          0          0          0
3               258.8          0          0          0          0          0
4               258.8          0          0          0          0          0

   …  department_Human Services  department_IT  department_Law and Reg  \
```

```
0  …                    0          0                0
1  …                    0          0                0
2  …                    0          0                0
3  …                    0          0                0
4  …                    0          0                0

   department_Libraries, Arts, Science, Museums  department_Parks  \
0                                             0                 1
1                                             0                 0
2                                             0                 0
3                                             0                 0
4                                             0                 0

   department_Police  department_Port  department_Public Works  city_id_2  \
0                  0                0                        0          1
1                  0                0                        0          1
2                  0                0                        0          1
3                  0                0                        0          1
4                  0                0                        1          1

   city_id_3
0          0
1          0
2          0
3          0
4          0

[5 rows x 26 columns]
```

[8]:
```python
# histograms
df4[[ 'base_salary' , 'overtime',        'irregular_cash', 'total_benefits']].
↪hist(grid=False, figsize=(18, 12))
plt.show()
```



0.7 Modeling

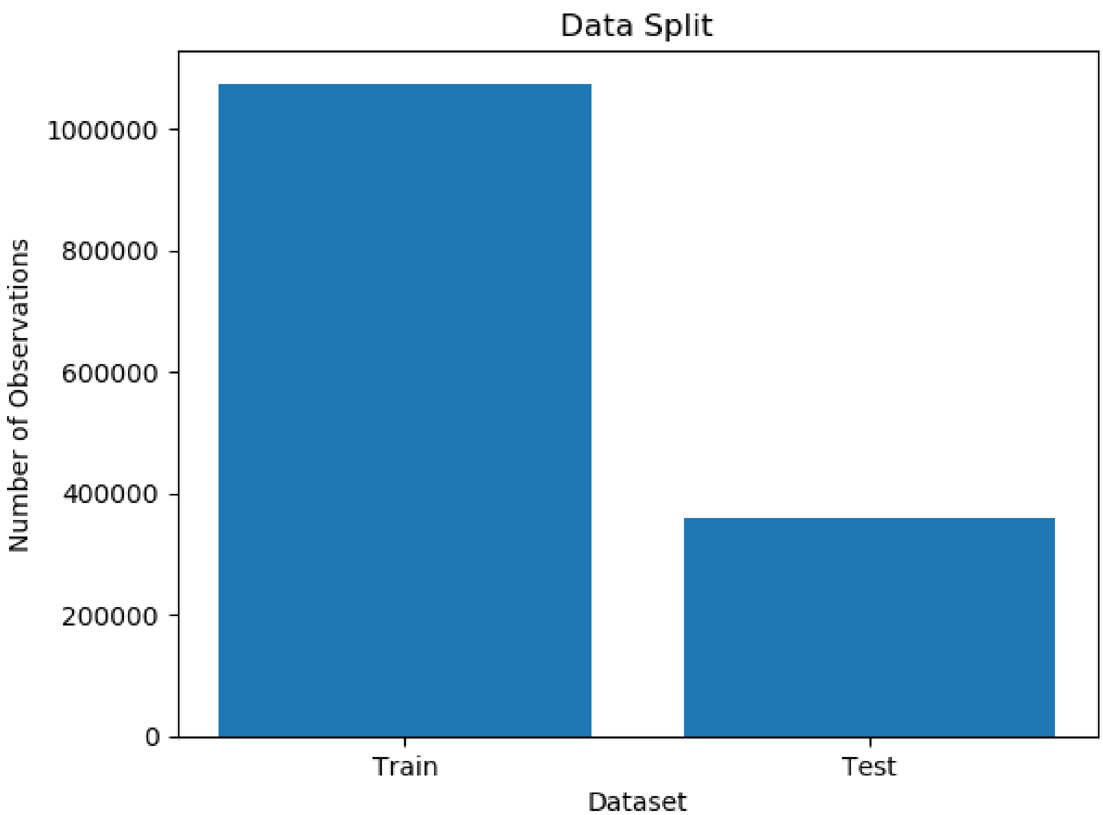
[86]:
```python
#set predictor and target dataframes
y = df4[['total_benefits']]
X= df4.drop(['total_benefits'], axis=1)
```

train/test split

[87]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler, PowerTransformer, RobustScaler

# Split all data into 75% train and 25% holdout
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=508)

# Convert target dataframe  to arrays
y_train2 = np.array(y_train)
y_test2 = np.array(y_test)
```

[11]:
```python
#show train, validation and test set chart
labels = ["Train",  "Test"]
sizes = [len(X_train.index),  len(X_test.index)]
```

```
plt.bar(labels, sizes)
plt.title("Data Split")
plt.xlabel("Dataset")
plt.ylabel("Number of Observations")
plt.figure(figsize=(8, 6))
plt.show()
```



<Figure size 800x600 with 0 Axes>

Normalize the data

[88]:
```
# Select the two columns you want to transform
cols_to_transform = [ 'base_salary' , 'overtime', 'irregular_cash']

# Initialize PowerTransformer and fit on the selected columns
X_train_fit = RobustScaler().fit(X_train[cols_to_transform ])

# Transform the selected columns for both training and testing set
X_train[cols_to_transform] = X_train_fit.transform(X_train[cols_to_transform])
X_test[cols_to_transform] = X_train_fit.transform(X_test[cols_to_transform])
```

–> Other methods such as miniscaler, min-max scaler, log transformation were also tried, not much improve in the results

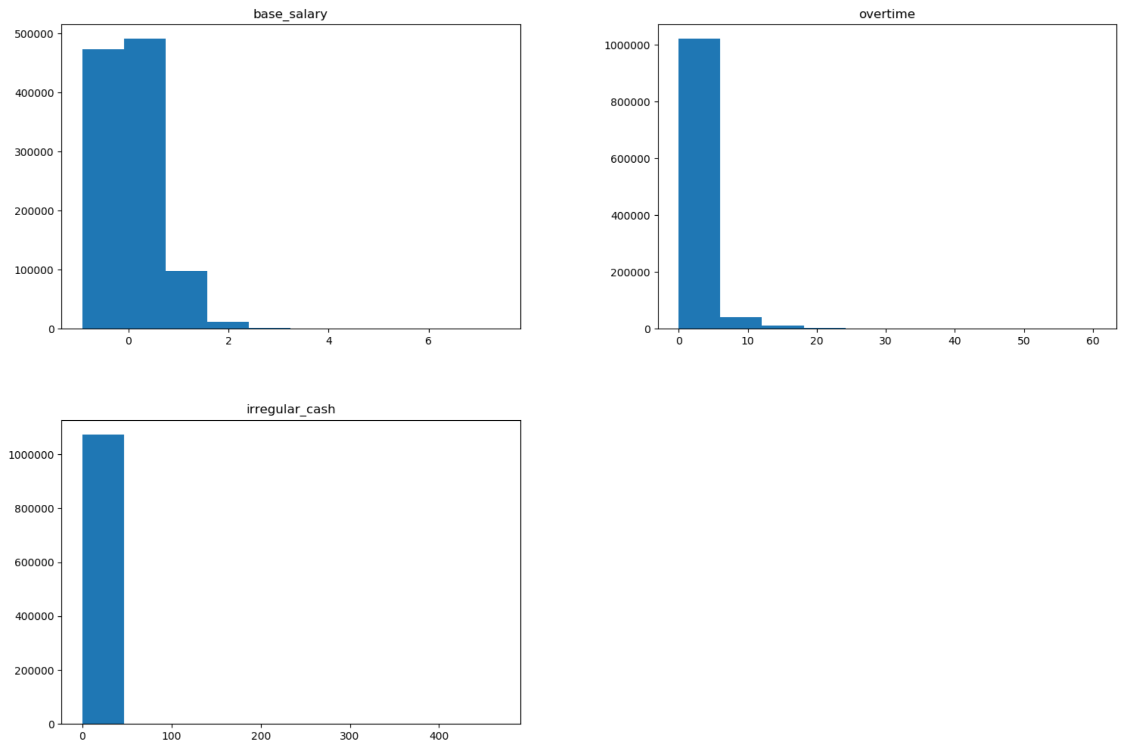
[15]:
```
X_train.head()
```

[15]:
```
         base_salary  overtime  irregular_cash  annual_average_cpi  year_2014  \
1241779    -0.635434 -0.015092        0.007521               251.1          0
1396548     0.665480 -0.015092       -0.135505               236.7          1
647017      0.317651  1.297260        0.920171               258.8          0
462797     -0.539971 -0.015092       -0.262128               240.0          0
966941      0.698426  4.337990        2.425067               240.0          0

         year_2015  year_2016  year_2017  year_2018  year_2019  …  \
1241779          0          0          0          1          0  …
1396548          0          0          0          0          0  …
647017           0          0          0          0          0  …
462797           0          1          0          0          0  …
966941           0          1          0          0          0  …

         department_Human Services  department_IT  department_Law and Reg  \
1241779                          0              0                       0
1396548                          0              1                       0
647017                           0              0                       0
462797                           0              0                       0
966941                           0              0                       0

         department_Libraries, Arts, Science, Museums  department_Parks  \
1241779                                             0                 0
1396548                                             0                 0
647017                                              0                 0
462797                                              0                 0
966941                                              0                 0

         department_Police  department_Port  department_Public Works  \
1241779                  0                0                        0
1396548                  0                0                        0
647017                   0                1                        0
462797                   0                0                        0
966941                   1                0                        0

         city_id_2  city_id_3
1241779          1          0
1396548          0          1
647017           1          0
462797           1          0
966941           1          0

[5 rows x 25 columns]
```

[16]:
```
X_train.shape
```

[16]: (1074602, 25)

[17]:
```
# histograms
X_train[[ 'base_salary' , 'overtime',         'irregular_cash']].
↪hist(grid=False, figsize=(18, 12))
plt.show()
```



0.7.1 Build pipeline

[89]:
```
import pandas as pd
import numpy as np
import boto3
import sagemaker
from sagemaker import get_execution_role
from sagemaker.sklearn.estimator import SKLearn
from sagemaker.tuner import IntegerParameter, ContinuousParameter,␣
↪HyperparameterTuner, CategoricalParameter
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor
from xgboost.sklearn import XGBRegressor
```

[92]:
```
from sklearn.metrics import mean_squared_error
#Define model function
def skl_reg_model(train_x=None,
                  train_y=None,
                  val_x=None,
                  val_y=None,
                  skl_model=None,
                  grid=None,
                  cv=5):

    if grid is None:
        model_fit = skl_model.fit(train_x, train_y)
    else:
        model_gridcv_fit = GridSearchCV(skl_model, grid, cv=cv).fit(train_x,␣
 ↪train_y)
        model_fit = model_gridcv_fit.best_estimator_
        print(f'Best CV grid parameters for {skl_model}: {model_gridcv_fit.
 ↪best_params_}')
    #performance on train set
    print('Training set')
    y_train_pred = model_fit.predict(train_x)
    print(f'RMSE = {np.sqrt(mean_squared_error(train_y, y_train_pred))}')
    print(f'R^2 score = {model_fit.score(train_x, train_y)}')
    #performance on test set
    print('Val/Test set')
    y_val_pred = model_fit.predict(val_x)
    print(f'RMSE =  {np.sqrt(mean_squared_error(val_y, y_val_pred))}')
    print(f'R^2 score = {model_fit.score(val_x, val_y)}')

    return model_fit
```

0.7.2 Multiple Linear Regression

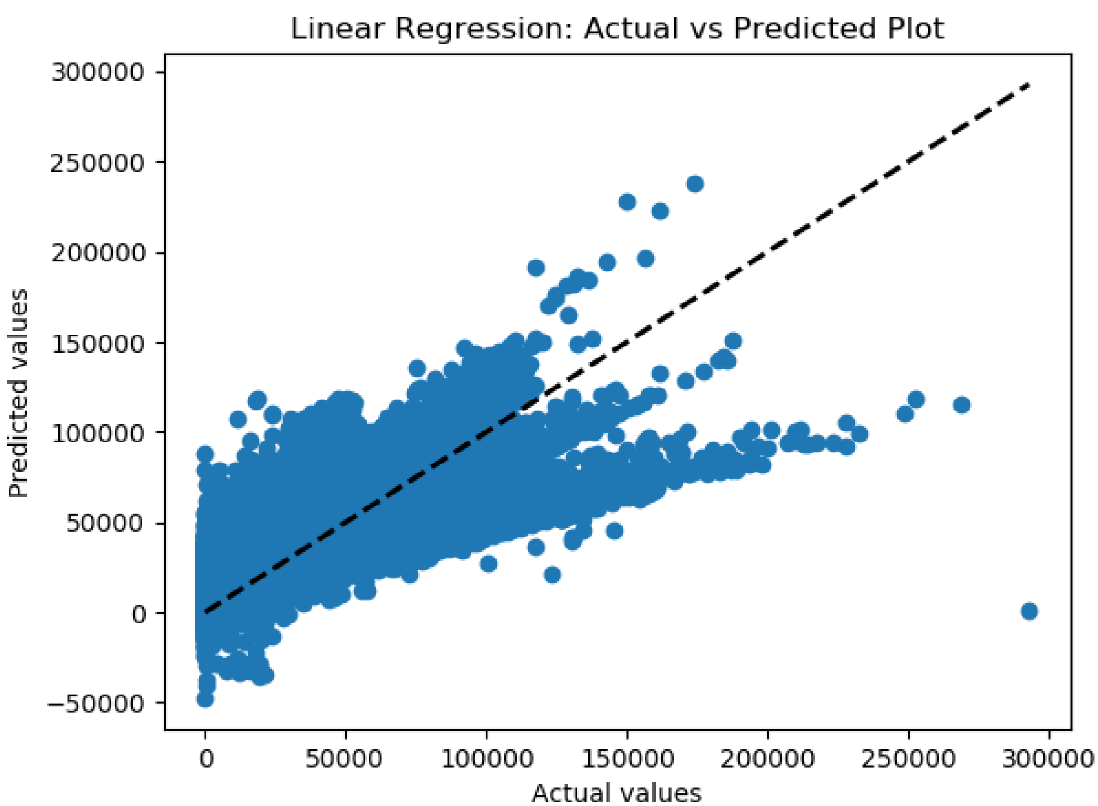
[29]:
```
lr = LinearRegression()
xgb_fit = skl_reg_model(train_x=X_train,
                                 train_y=y_train2,
                                 val_x= X_test,
                                 val_y=y_test2,
                                 skl_model=lr,
                                 grid=None)
```

```
Training set
RMSE = 11262.942500672569
R^2 score = 0.7785075648636522
Val/Test set
RMSE =  11250.913003130454
R^2 score = 0.779924519366728
```

[30]:
```
# Print the parameters (coefficients) of the model
print("Coefficients:", lr.coef_)
print("Intercept:", lr.intercept_)
```

```
Coefficients: [[ 3.11449401e+04  5.82492406e+02 -8.28762225e+02 -1.67377864e+01
   1.30217005e+03  1.05100506e+03 -2.07554781e+02 -1.37142182e+03
  -1.21041755e+03 -1.03996015e+03  9.62522728e+01  1.36949065e+03
   9.08335527e+03  1.33798498e+03  1.58862289e+03  2.34578762e+03
   7.78327829e+02  2.83590313e+03 -9.51828296e+02 -1.51662261e+03
   9.73754427e+03  4.25806004e+03 -2.55623778e+02 -4.27521374e+03
  -3.68375816e+03]]
Intercept: [38194.70557889]
```

[35]:
```
#Plot predicted value and the actual test data
y_val_pred = xgb_fit.predict(X_test)
plt.scatter(y_test2, y_val_pred )
plt.plot([y_test2.min(), y_test2.max()], [y_test2.min(), y_test2.max()], 'k--',␣
 ↪lw=2)
plt.xlabel('Actual values')
plt.ylabel('Predicted values')
plt.title('Linear Regression: Actual vs Predicted Plot')
plt.show()
```

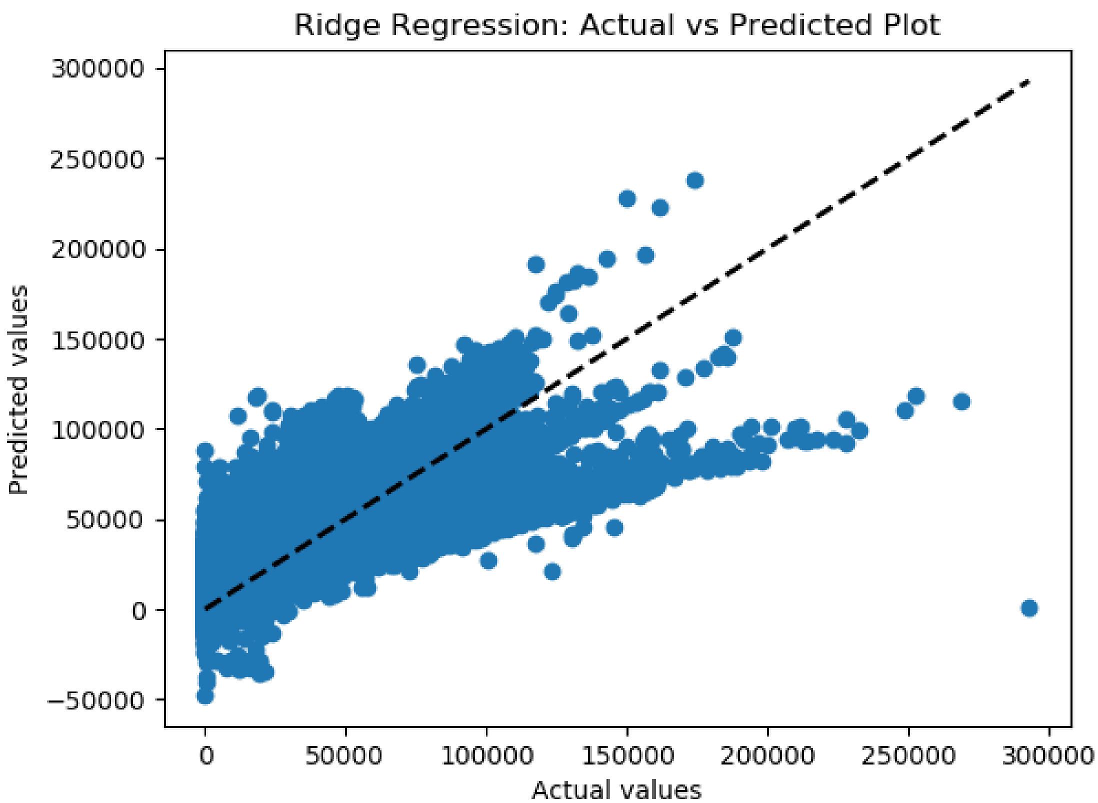


0.7.3 Ridge Regression

[36]:
```
#Fit a Ridge regression model
ridge = Ridge()
ridge_grid = {'alpha': [0.01, 0.1, 0.2, 0.5, 1, 10, 100]}

ridge_fit = skl_reg_model(train_x=X_train,
                          train_y=y_train2,
                          val_x= X_test,
                          val_y=y_test2,
                          skl_model=ridge,
                          grid=ridge_grid)
```

```
Best CV grid parameters for Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001): {'alpha':
10}
Training set
RMSE = 11262.942605068081
R^2 score = 0.778507560757653
Val/Test set
RMSE =  11250.914544747788
R^2 score = 0.7799244590565649
```

[37]:
```
#Plot predicted value and the actual test data
y_val_pred = ridge_fit.predict(X_test)
plt.scatter(y_test2, y_val_pred)
plt.plot([y_test2.min(), y_test2.max()], [y_test2.min(), y_test2.max()], 'k--',␣
 ↪lw=2)
plt.xlabel('Actual values')
plt.ylabel('Predicted values')
plt.title('Ridge Regression: Actual vs Predicted Plot')
plt.show()
```



0.7.4 XGBoost

[14]:
```
xgb = XGBRegressor(n_estimators=100)
xgb_grid = {
    'learning_rate': [0.01, 0.1, 0.5],
    'max_depth': [3, 5, 7],
    'gamma': [0, 0.1, 0.5]}

xgb_fit = skl_reg_model(train_x=X_train,
                                 train_y=y_train2,
                                 val_x= X_test,
                                 val_y=y_test2,
                                 skl_model=xgb,
                                 grid=xgb_grid)
```

```
Best CV grid parameters for XGBRegressor(base_score=None, booster=None,
callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, gamma=None,
             gpu_id=None, grow_policy=None, importance_type=None,
```

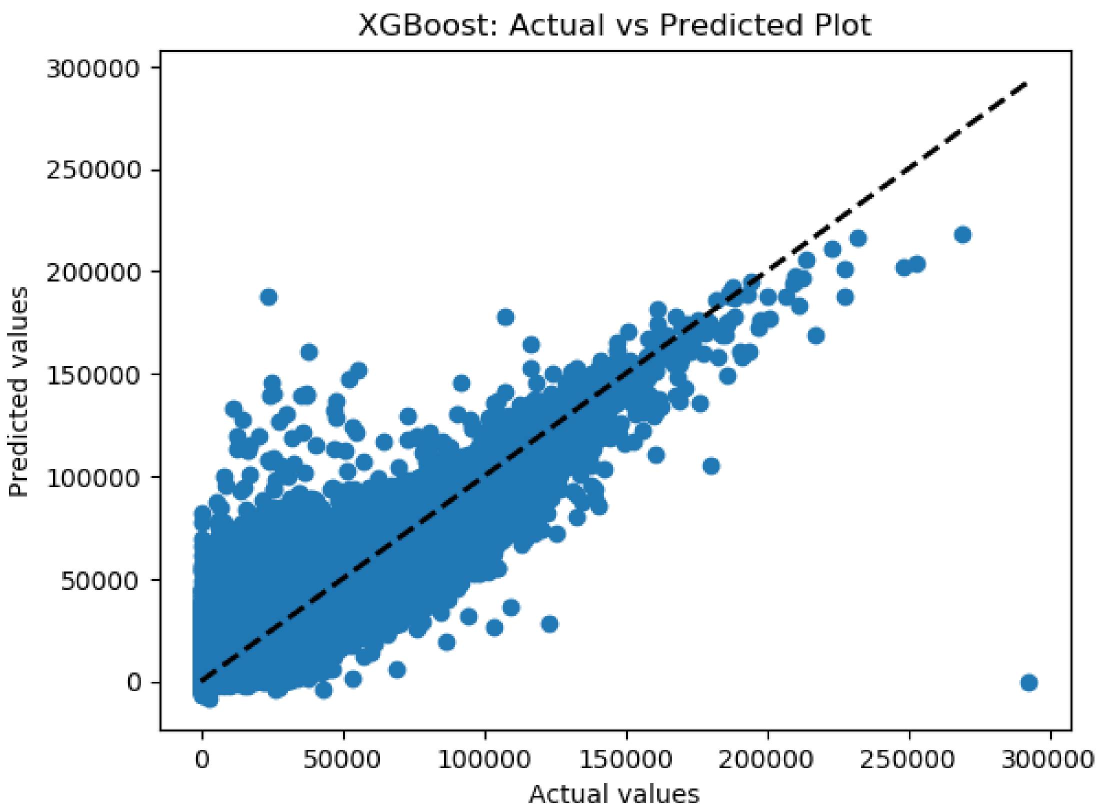
```
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, n_estimators=100, n_jobs=None,
             num_parallel_tree=None, objective='reg:squarederror',
             predictor=None, random_state=None, reg_alpha=None, …): {'gamma':
0, 'learning_rate': 0.5, 'max_depth': 7}
Training set
RMSE = 6148.2111224562805
R^2 score = 0.9339986667595346
Val/Test set
RMSE =  6416.296054883127
R^2 score = 0.9284244135575569
```

[39]:
```
best_xgb_fit = XGBRegressor(n_estimators=100, learning_rate = 0.5, max_depth=7,␣
 ↪gamma=0). fit(X_train, y_train2)
y_val_pred = best_xgb_fit.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test2, y_val_pred))
```

[40]:
```
rmse
```

[40]:
```
6416.296054883127
```

[41]:
```
#Plot predicted value and the actual test data
y_val_pred = best_xgb_fit.predict(X_test)
plt.scatter(y_test2, y_val_pred)
plt.plot([y_test2.min(), y_test2.max()], [y_test2.min(), y_test2.max()], 'k--',␣
 ↪lw=2)
plt.xlabel('Actual values')
plt.ylabel('Predicted values')
plt.title('XGBoost: Actual vs Predicted Plot')
plt.show()
```



0.7.5 Random Forest

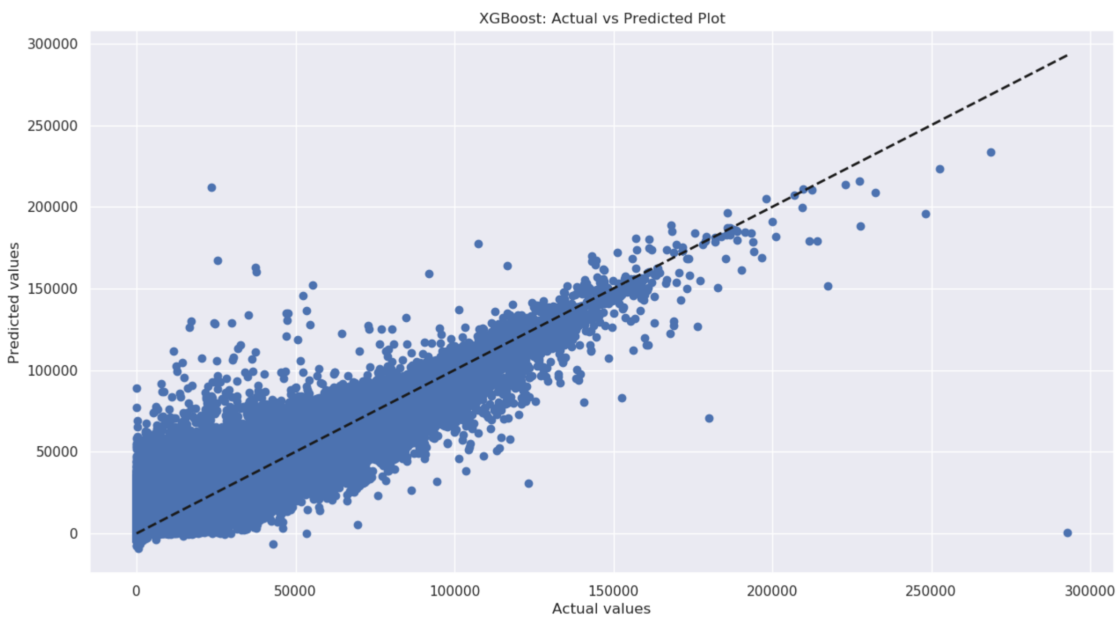
[93]:
```
#Random forest with tuning
rf = RandomForestRegressor(n_estimators=300)
rf_grid = { 'max_depth': [3,5,7]}

rf_fit = skl_reg_model(train_x=X_train,
                                  train_y=y_train2,
                                  val_x= X_test,
                                  val_y=y_test2,
                                  skl_model=rf,
                                  grid=rf_grid)
```

```
Best CV grid parameters for RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=300, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False):
```

```
{'max_depth': 7}
Training set
RMSE = 8955.297361353529
R^2 score = 0.8599719031158484
Val/Test set
RMSE =  8972.501920955461
R^2 score = 0.8600337761649571
```

[96]:
```
rf_fit = XGBRegressor(n_estimators=300, max_depth=7). fit(X_train, y_train2)
y_val_pred = rf_fit.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test2, y_val_pred))
```

[97]:
```
#Plot predicted value and the actual test data
y_val_pred = rf_fit.predict(X_test)
plt.scatter(y_test2, y_val_pred)
plt.plot([y_test2.min(), y_test2.max()], [y_test2.min(), y_test2.max()], 'k--',␣
 ↪lw=2)
plt.xlabel('Actual values')
plt.ylabel('Predicted values')
plt.title('XGBoost: Actual vs Predicted Plot')
plt.show()
```



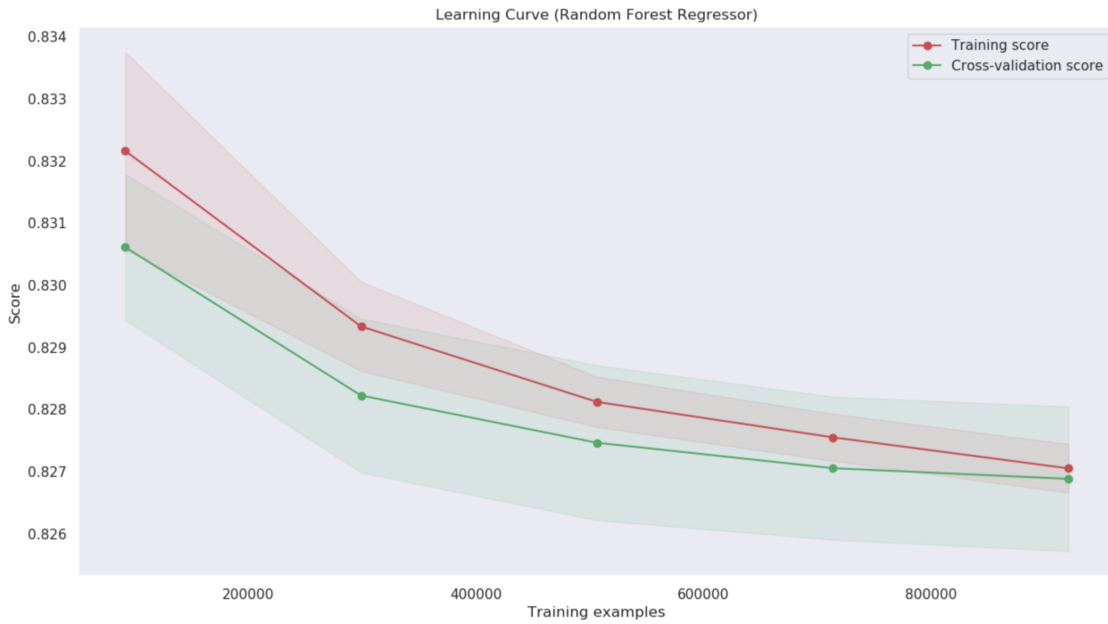
Learner curve plot

[85]:
```
from sklearn.model_selection import learning_curve
```

```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,␣
↪n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")
    plt.legend(loc="best")
    return plt


title = "Learning Curve (Random Forest Regressor)"
cv = 5 # number of cross-validation folds
plot_learning_curve(rf, title, X_train2, y_train2.ravel(), cv=cv)
```

[85]:
```
<module 'matplotlib.pyplot' from '/opt/conda/lib/python3.7/site-
packages/matplotlib/pyplot.py'>
```

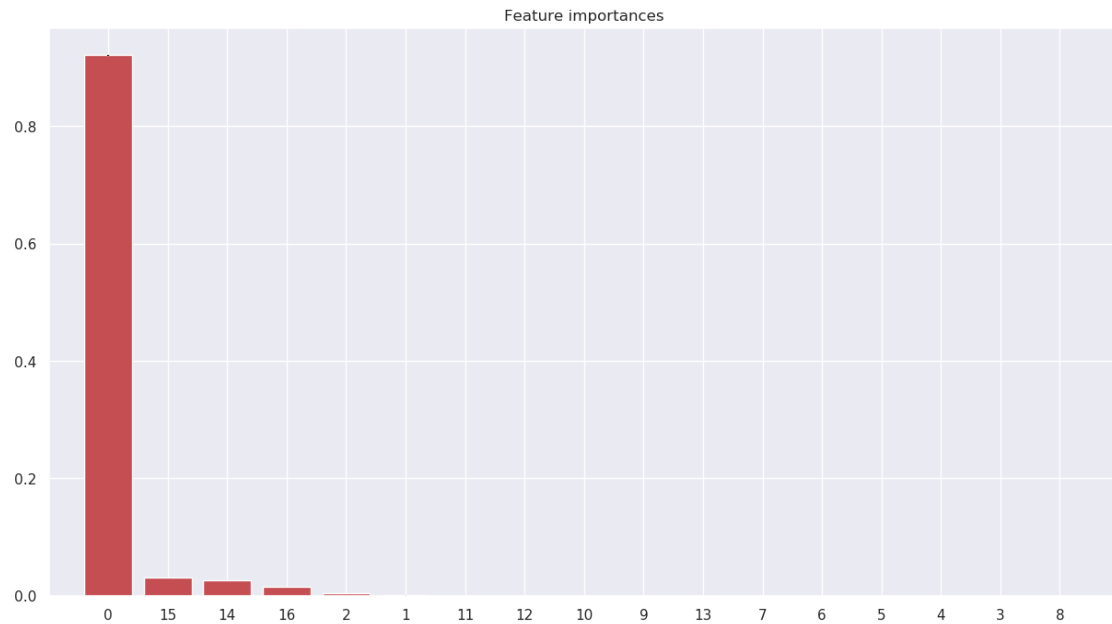
[86]:
```
plt.show()
```



Feature importance

–> Training result seems to not much improve after the sample size reach around 500K-600K instances

[87]:
```python
#Feature importance
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt

# # train a Random Forest Regressor model
# rf = RandomForestRegressor(n_estimators=100, random_state=42)
# rf.fit(X_train, y_train)

# plot feature importances
importances = rf.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

plt.figure()
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```



–> Base_salary showed to be the most important feature. In fact, it is explain more than 90% of the results.

Examine the base_salary alone in the training

[112]:
```
# Create the scatter plot
plt.scatter(df4['base_salary'], df4['total_benefits'])
plt.xlabel('Salary')
plt.ylabel('Total Benefits')
plt.title('Salary vs Total Benefits Scatter Plot')
plt.show()
```

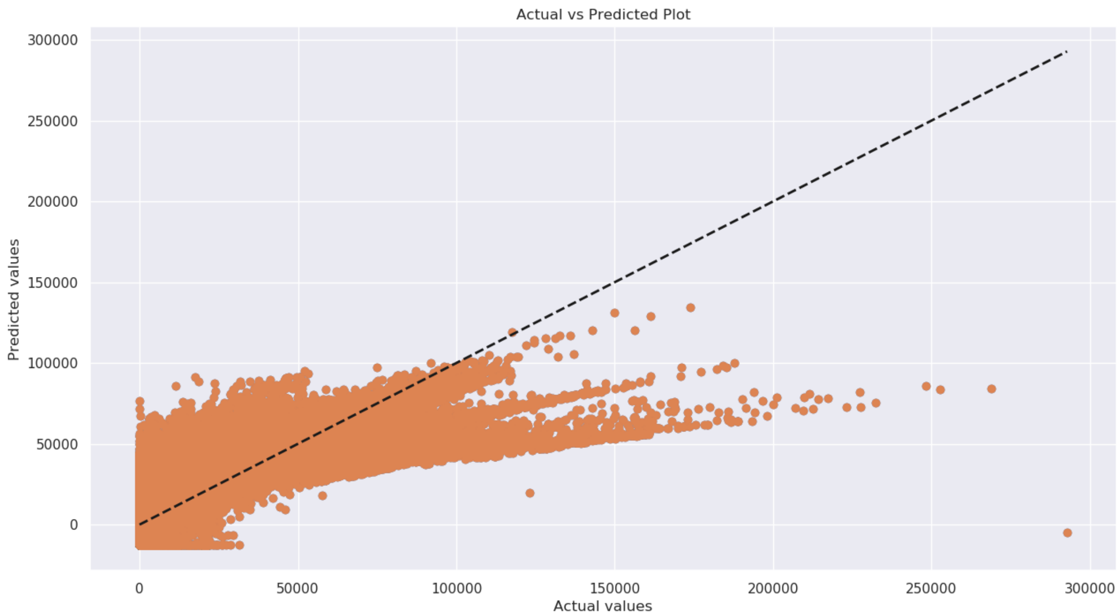
[100]:
```
#Linear regression
lr2= LinearRegression().fit(np.array(X_train['base_salary']).reshape(-1, 1),␣
↪y_train2)
```

[101]:
```
# make predictions on the test data
y_pred = lr2.predict(np.array(X_test['base_salary']).reshape(-1, 1))

rmse = np.sqrt(mean_squared_error(y_test2, y_pred))
```

[103]:
```
# calculate the RMSE between the predicted and true values
#rmse = np.sqrt(mean_squared_error(y_val2, y_pred))
mae = mean_absolute_error(y_test2, y_pred)
mae
```

[103]: 7899.037006717493

[105]:
```
plt.scatter(y_test2, y_pred)
plt.plot([y_test2.min(), y_test2.max()], [y_test2.min(), y_test2.max()], 'k--',␣
↪lw=2)
plt.xlabel('Actual values')
plt.ylabel('Predicted values')
plt.title('Actual vs Predicted Plot')
plt.show()
```



–> model run on only base_salary feature can predict most of the target. Other features seem to be not very useful. In the future, more feature need to be explore. Job_title could be a great one if we could manage to condense values into relevant groups.

Shutting down your kernel for this notebook to release resources.

[ ]:
```
%%html

<p><b>Shutting down your kernel for this notebook to release resources.</b></p>
<button class="sm-command-button" data-commandlinker-command="kernelmenu:
↪shutdown" style="display:none;">Shutdown Kernel</button>

<script>
try {
    els = document.getElementsByClassName("sm-command-button");
    els[0].click();
}
catch(err) {
    // NoOp
}
</script>
```

Shutting down all checkpoint and kernels for all notebooks

[ ]:
```
%%javascript
```

```javascript
try {
    Jupyter.notebook.save_checkpoint();
    Jupyter.notebook.session.delete();
}
catch(err) {
    // NoOp
}
```

eda_harini

April 17, 2023

[1]:
```
import gzip
# Import dependences
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn import preprocessing
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split, KFold, cross_val_score
import sklearn.metrics as metrics
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪precision_score,recall_score, f1_score
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import NearestNeighbors,␣
↪KNeighborsClassifier,KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.svm import SVR
from dmba import AIC_score, adjusted_r2_score, BIC_score, stepwise_selection
import math
import operator
from prettytable import PrettyTable
warnings.filterwarnings("ignore")
%matplotlib inline
```

[2]:
```
with gzip.open('compensation_cpi.csv.gz', 'rb') as f:
    df = pd.read_csv(f)
```

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 1466589 entries, 0 to 1466588
Data columns (total 15 columns):
 #   Column               Non-Null Count    Dtype
---  ------               --------------    -----
 0   year                 1466589 non-null  int64
 1   department           1466587 non-null  object
 2   job_title            1466043 non-null  object
 3   base_salary          1465890 non-null  float64
 4   overtime             1433207 non-null  float64
 5   irregular_cash       1466155 non-null  float64
 6   total_cash           1466589 non-null  float64
 7   retirement           1466589 non-null  float64
 8   health               1447602 non-null  float64
 9   other_benefits       830557 non-null   float64
 10  total_benefits       1466589 non-null  float64
 11  total_compensation   1466589 non-null  float64
 12  city_id              1466589 non-null  int64
 13  annual_average_cpi   1466589 non-null  float64
 14  inflation_rate       1466589 non-null  float64
dtypes: float64(11), int64(2), object(2)
memory usage: 167.8+ MB
```

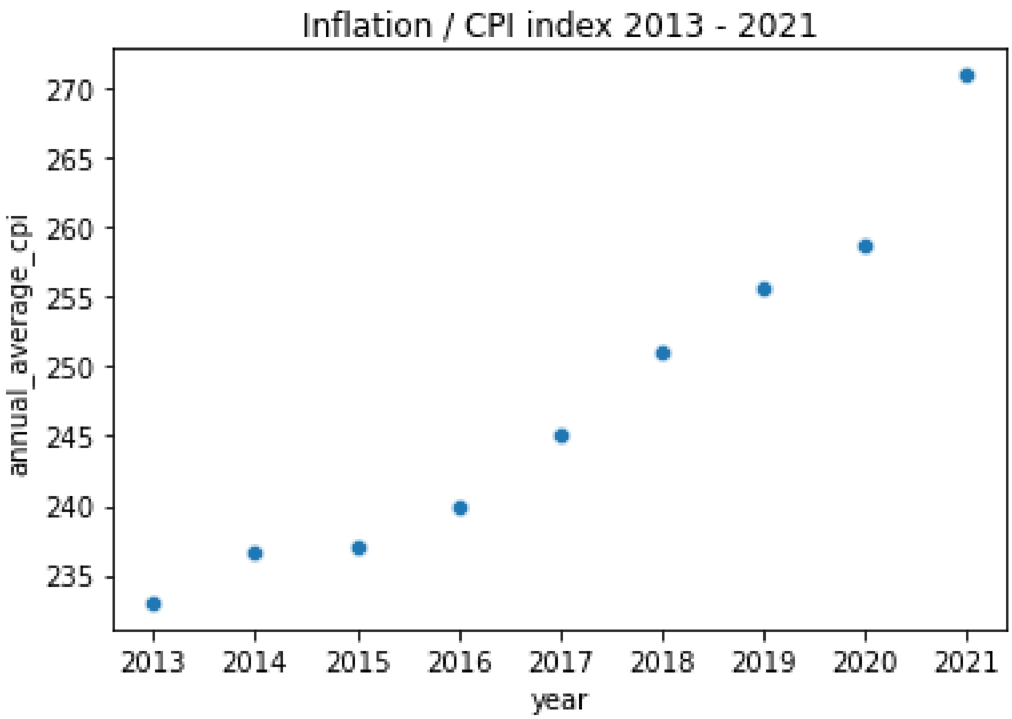
[4]:
```
df.head(3)
```

[4]:
```
   year                      department         job_title  base_salary  \
0  2020  Recreation And Park Commission    Camp Assistant      5257.50
1  2020                       Registrar      Junior Clerk      7699.19
2  2020                       Registrar      Junior Clerk      2619.15

   overtime  irregular_cash  total_cash  retirement  health  other_benefits  \
0       0.0          139.32     5396.82         0.0     0.0          418.88
1    1916.9            0.00     9616.09         0.0     0.0          746.36
2     930.5            0.00     3549.65         0.0     0.0          275.51

   total_benefits  total_compensation  city_id  annual_average_cpi  \
0          418.88             5815.70        2               258.8
1          746.36            10362.45        2               258.8
2          275.51             3825.16        2               258.8

   inflation_rate
0             1.2
1             1.2
2             1.2
```

[5]:
```
df[df['department'].str.contains('District') == True]['department'].unique()
```

[5]:
```
array(['District Attorney', 'DAT District Attorney',
       'Council - District 7', 'Council - District 2',
       'Council - District 10', 'Council - District 8',
       'Council - District 3', 'Council - District 5',
       'Council - District 1', 'Council - District 9',
       'Council - District 4', 'Council - District 6', 'Parks District 8'],
      dtype=object)
```

[6]:
```
sns.scatterplot(data=df,x='year',y="annual_average_cpi")
plt.title("Inflation / CPI index 2013 - 2021");
```



[7]:
```
#we have two missing department names but they are both police
df[df['department'].isna()]
```

[7]:
```
        year department         job_title  base_salary  overtime  \
139761  2017        NaN    Sheriff's Cadet     49630.50  15016.51
430579  2017        NaN  Police Officer 2    116189.62  40990.09

        irregular_cash  total_cash  retirement    health  other_benefits  \
139761         3197.52    67844.53    10619.27  12779.88         4796.56
430579         2260.08   159439.80    20076.66  14515.01         2724.05

        total_benefits  total_compensation  city_id  annual_average_cpi  \
```

```
139761        28195.71              96040.24         2                 245.1
430579        37315.72             196755.52         2                 245.1

        inflation_rate
139761             2.1
430579             2.1
```

[8]:
```
#fill in the missing police department names
df['department'].fillna('Police',inplace=True)
```

[9]:
```
df['department'].value_counts()
```

[9]:
```
POLICE                          136819
WATER AND POWER                 109906
Public Health                    95725
RECREATION AND PARKS             84680
DPH Public Health                73093
                                  …
Airport-Custodians                   1
Police-Crisis Management             1
Police-TABS                          1
Attorney-Part Time                   1
DOT/Pavement Maint Southeast         1
Name: department, Length: 550, dtype: int64
```

0.0.1 Condensing Department Names

[10]:
```
def replace_text(text):
    if pd.isna(text) or text is None:
        return text
    elif target_word.lower() in text.lower():
        return new_word
    else:
        return text
```

[11]:
```
target_word= "Police"
new_word= "Police"
df['department'] = df['department'].apply(replace_text)
```

[12]:
```
dept_dict = {
    'Police': 'Police', 'Sheriff': 'Police', "Vcet" : "Police",
    "Fire" : "Emergency Management",    "Emergency" : "Emergency Management",
    "PW" : "Public Works",    "Public" : "Public Works",    "Water" : "Public␣
↪Works",    "DOT" : "Public Works",    "Transport" : "Public Works",
    "Plan" : "Public Works",    "Building" : "Public Works",    #"District" :␣
↪"Public Works",
```

```
    "PRNS" : "Parks",    "Recre" : "Parks",    "Zoo" : "Parks",    "Parks" :␣
↪"Parks",    "Arena" : "Parks",
    "City" : "City Mgmt",    "Convention" : "City Mgmt",    "Neighbor" : "City␣
↪Mgmt",    "Election" : "City Mgmt",    "Council" : "City Mgmt",
    "CII" : "City Mgmt",    "Clerk" : "City Mgmt",    "Registrar" : "City␣
↪Mgmt",    "Housing" : "City Mgmt",    "Mayor" : "City Mgmt",    "rda" :␣
↪"City Mgmt",
    "Airport" : "Airport",    "Airside" : "Airport",
    "Finance" : "Finance",    "Auditor" : "Finance",    "Assessor" : "Finance",␣
↪   "Controller" : "Finance",    "Tax" : "Finance", "Treasure" : "Finance",
    "Board" : "Law and Reg",    "Attorney" : "Law and Reg",    "Court" : "Law␣
↪and Reg",
    "Ethics" : "Law and Reg",    "Probation" : "Law and Reg",    "Regulation" :␣
↪"Law and Reg",
    "prt" : "Port",    "port" : "Port", "Harbor" : "Port",
    "Human" : "Human Services",    "Retire" : "Human Services",    "Child" :␣
↪"Human Services",    "Service" : "Human Services",
    "Personnel" : "Human Services",    "Aging" : "Human Services",    "Women" :␣
↪"Human Services",    "Pension" : "Human Services",
    "Disability" : "Human Services",    "Families" : "Human Services", "Youth" :
↪ "Human Services",
    "ESD" : "Human Services",    "Employee" : "Human Services",
    "Info" : "IT",    "Tech" : "IT",
    "Envi" : "Energy, Env, Economy",    "Energy" : "Energy, Env, Economy",    ␣
↪"Power" : "Energy, Env, Economy", "Econ" : "Energy, Env, Economy",
    "Science" : "Libraries, Arts, Science, Museums",    "Librar" : "Libraries,␣
↪Arts, Science, Museums",    "Museum" : "Libraries, Arts, Science, Museums",
    "Memorial" : "Libraries, Arts, Science, Museums",    "Monument" :␣
↪"Libraries, Arts, Science, Museums",    "Arts" : "Libraries, Arts, Science,␣
↪Museums",
    "Cultur" : "Libraries, Arts, Science, Museums", "Art Commission" :␣
↪"Libraries, Arts, Science, Museums"
}
```

[13]:
```
for key in dept_dict:
    target_word= key
    new_word= dept_dict[key]
    df['department'] = df['department'].apply(replace_text)
```

[14]:
```
df['department'].value_counts(normalize=True)
```

[14]:
```
Public Works                     0.384807
Police                           0.159051
Port                             0.098913
Parks                            0.097807
Human Services                   0.085051
```

```
Emergency Management                    0.057883
City Mgmt                               0.046575
Libraries, Arts, Science, Museums       0.022667
Law and Reg                             0.019740
Finance                                 0.014328
IT                                      0.008094
Energy, Env, Economy                    0.005083
Name: department, dtype: float64
```

0.0.2 Adjusting all numbers to present-day CPI

[15]:
```
df['base_salary'] = df['base_salary'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['overtime'] = df['overtime'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['irregular_cash'] = df['irregular_cash'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['total_cash'] = df['total_cash'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])

df['retirement'] = df['retirement'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['health'] = df['health'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['other_benefits'] = df['other_benefits'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['total_benefits'] = df['total_benefits'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['total_compensation'] = df['total_compensation'] * (df['annual_average_cpi'].
 ↪max() / df['annual_average_cpi'])
```

[16]:
```
df.head()
```

[16]:
```
   year    department        job_title    base_salary     overtime  \
0  2020         Parks   Camp Assistant    5505.341963     0.000000
1  2020     City Mgmt     Junior Clerk    8062.134815  2007.263910
2  2020     City Mgmt     Junior Clerk    2742.618431   974.364374
3  2020     City Mgmt            Clerk    1958.802241   619.090495
4  2020  Public Works         Engineer  166298.647372     0.000000

   irregular_cash     total_cash   retirement        health  other_benefits  \
0      145.887635    5651.229598     0.000000      0.000000      438.626275
1        0.000000   10069.398725     0.000000      0.000000      781.543895
2        0.000000    3716.982805     0.000000      0.000000      288.497720
3        0.000000    2577.892736     0.000000      0.000000      200.077164
4     5944.554637  172243.202009  35106.269861  15799.729328    12219.283192
```

```
   total_benefits  total_compensation  city_id  annual_average_cpi  \
0      438.626275         6089.855873        2               258.8
1      781.543895        10850.942620        2               258.8
2      288.497720         4005.480526        2               258.8
3      200.077164         2777.969900        2               258.8
4    63125.282380       235368.484389        2               258.8

   inflation_rate
0             1.2
1             1.2
2             1.2
3             1.2
4             1.2
```

0.0.3 More cleaning

Changing integers for Year and City into categorical variables

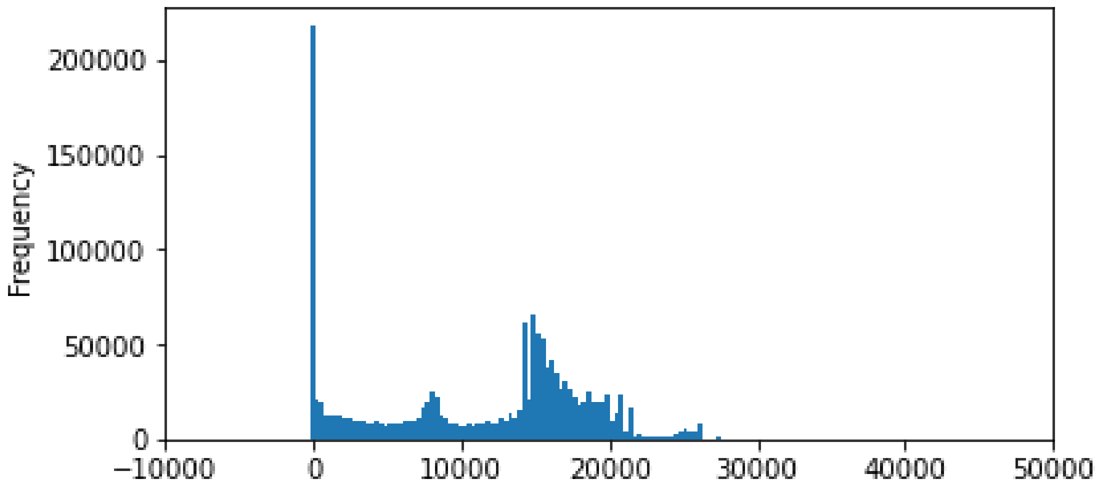
[17]:
```python
df['year'] = df['year'].astype('category')
df['city_id'] = df['city_id'].astype('category')
```

[18]:
```python
df.isna().sum()
```

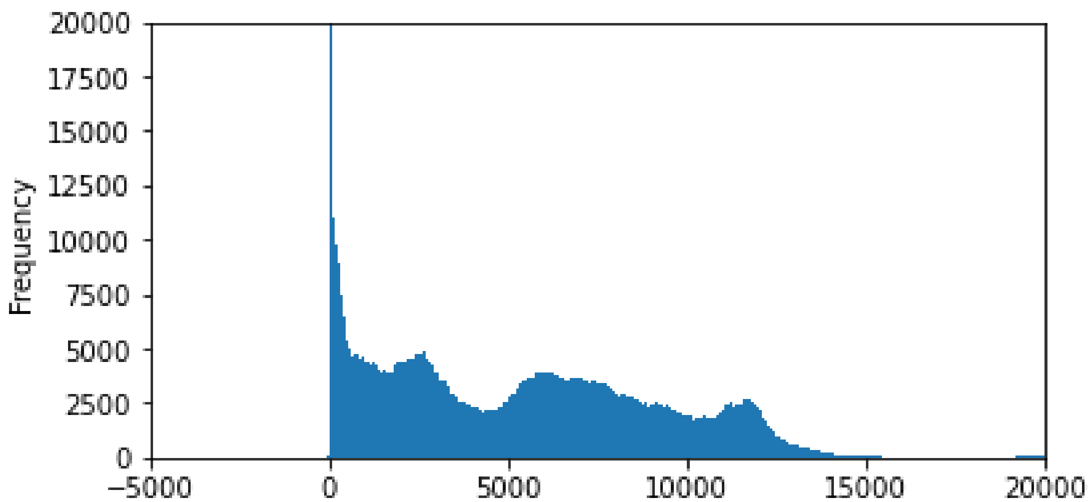
[18]:
```
year                       0
department                 0
job_title                546
base_salary              699
overtime               33382
irregular_cash           434
total_cash                 0
retirement                 0
health                 18987
other_benefits        636032
total_benefits             0
total_compensation         0
city_id                    0
annual_average_cpi         0
inflation_rate             0
dtype: int64
```

I feel justified filling in null “health” and “other_benefits” values with 0 because it’s the single most common practice

[19]:
```python
plt.figure(figsize=(6,3))
plt.xlim(-10000,50000)
df['health'].plot.hist(bins=1000);
```



[20]:
```
plt.figure(figsize=(6,3))
plt.xlim(-5000,20000)
plt.ylim(0,20000)
df['other_benefits'].plot.hist(bins=1000);
```



[21]:
```
df['job_title'] = df['job_title'].fillna("Not disclosed")
df['overtime'] = df['overtime'].fillna(0)
df['irregular_cash'] = df['irregular_cash'].fillna(0)
df['health'] = df['health'].fillna(0)
df['other_benefits'] = df['other_benefits'].fillna(0)
```

```
df['base_salary'] = df['base_salary'].fillna((df['total_cash'] - df['overtime']␣
↪- df['irregular_cash']))

df.isna().sum()
```

[21]:
```
year                   0
department             0
job_title              0
base_salary            0
overtime               0
irregular_cash         0
total_cash             0
retirement             0
health                 0
other_benefits         0
total_benefits         0
total_compensation     0
city_id                0
annual_average_cpi     0
inflation_rate         0
dtype: int64
```

[22]:
```
len(df)
```

[22]:
```
1466589
```

[23]:
```
df = df[df['base_salary'] >= 0]
```

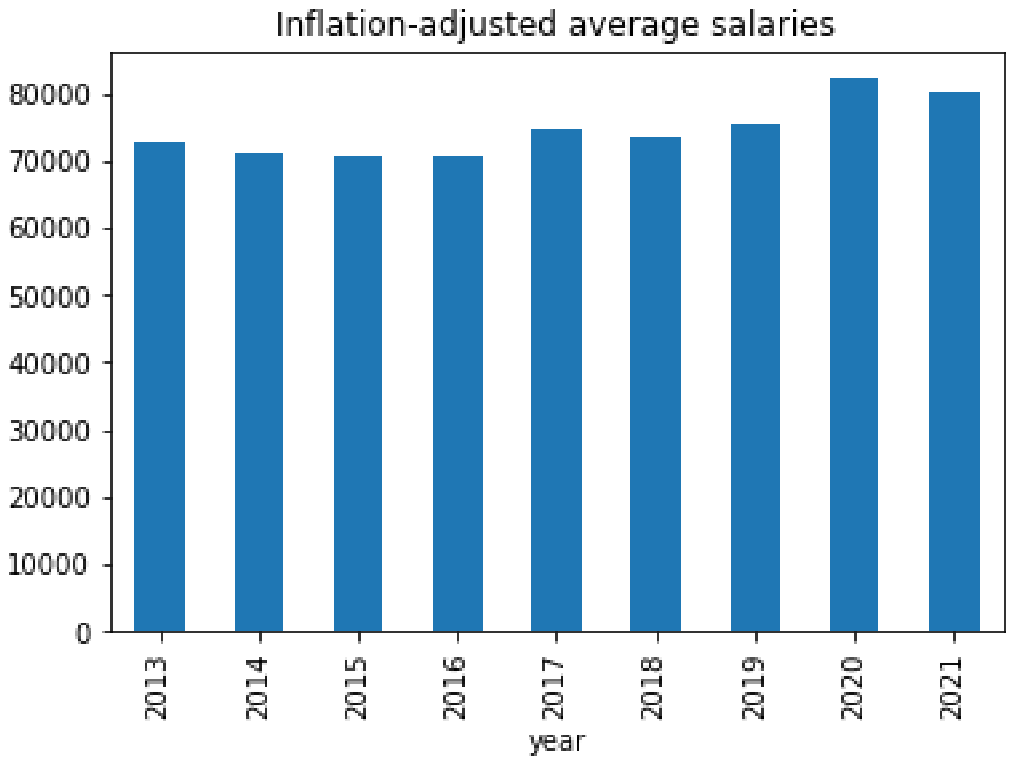
[24]:
```
df = df[df['total_compensation'] >= 0]
```

[25]:
```
len(df)
```

[25]:
```
1465857
```

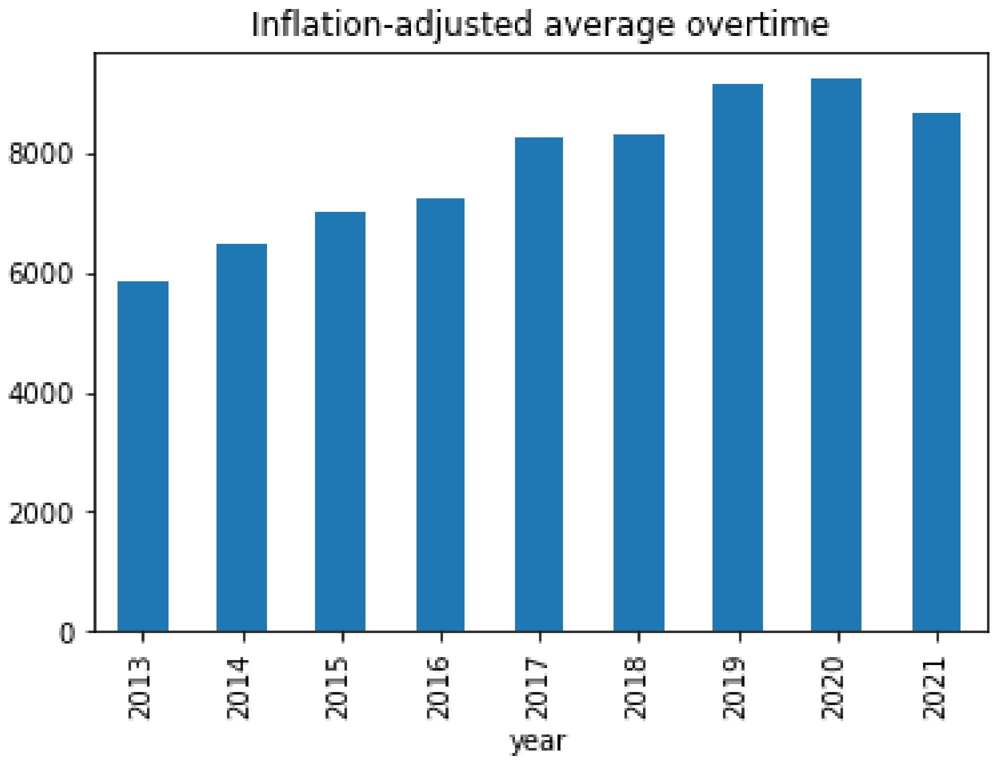
0.0.4 Some visuals

[26]:
```
df.groupby('year')['base_salary'].mean().plot.bar()
plt.title("Inflation-adjusted average salaries");
```

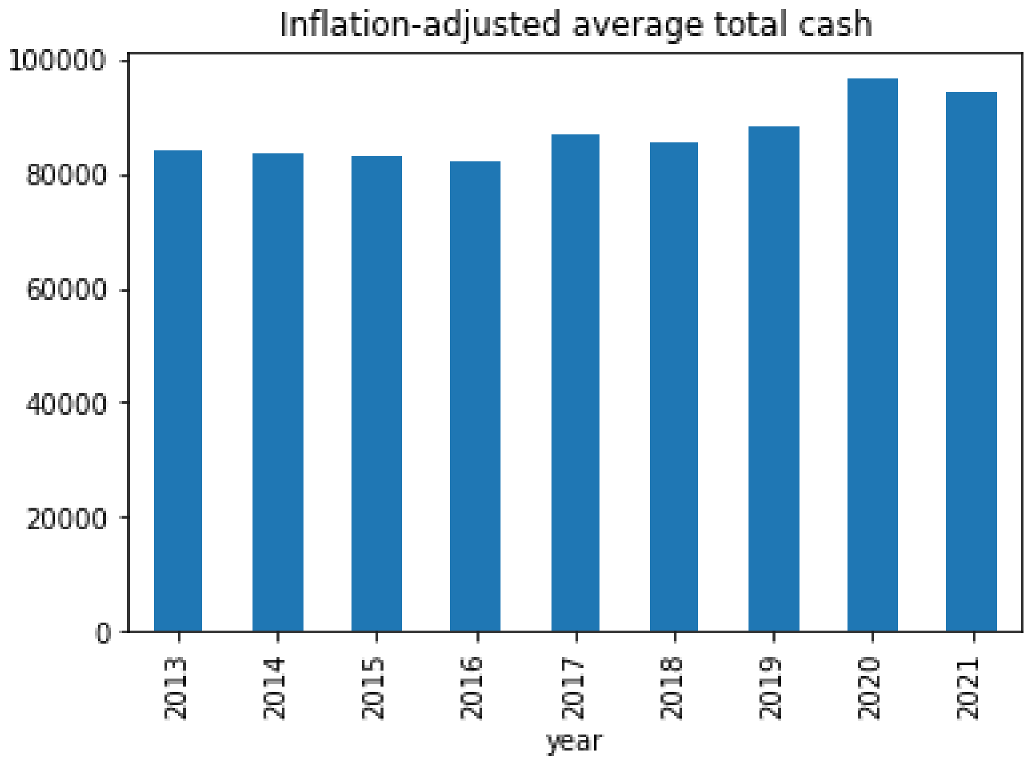


[27]:

```
df.groupby('year')['overtime'].mean().plot.bar()
plt.title("Inflation-adjusted average overtime");
```

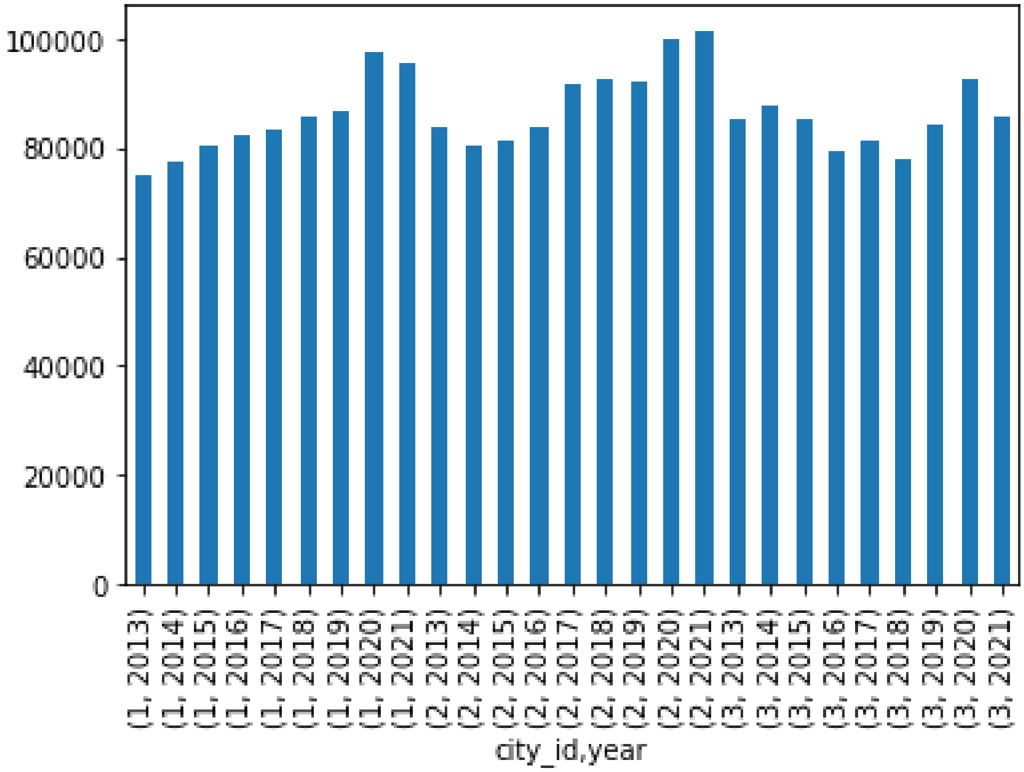


[28]:
```
df.groupby('year')['total_cash'].mean().plot.bar()
plt.title("Inflation-adjusted average total cash");
```



[29]:
```
df.groupby(['city_id','year'])['total_cash'].mean().plot.bar()
```

[29]: <AxesSubplot:xlabel='city_id,year'>



[30]:
```
df = pd.get_dummies(df,columns=['department'],drop_first=True)
```

[31]:
```
df_copy=df.drop(['job_title', 'total_cash', 'retirement', 'health',␣
↪'other_benefits','total_compensation','inflation_rate'], axis=1)
df_copy.head()
```

[31]:
```
   year     base_salary     overtime  irregular_cash  total_benefits city_id  \
0  2020     5505.341963     0.000000      145.887635      438.626275       2
1  2020     8062.134815  2007.263910        0.000000      781.543895       2
2  2020     2742.618431   974.364374        0.000000      288.497720       2
3  2020     1958.802241   619.090495        0.000000      200.077164       2
4  2020   166298.647372     0.000000     5944.554637    63125.282380       2

   annual_average_cpi  department_Emergency Management  \
0               258.8                                0
1               258.8                                0
2               258.8                                0
3               258.8                                0
4               258.8                                0

   department_Energy, Env, Economy  department_Finance  \
0                                0                   0
```

```
1                                0                   0
2                                0                   0
3                                0                   0
4                                0                   0

   department_Human Services  department_IT  department_Law and Reg  \
0                          0              0                       0
1                          0              0                       0
2                          0              0                       0
3                          0              0                       0
4                          0              0                       0

   department_Libraries, Arts, Science, Museums  department_Parks  \
0                                             0                 1
1                                             0                 0
2                                             0                 0
3                                             0                 0
4                                             0                 0

   department_Police  department_Port  department_Public Works
0                  0                0                       0
1                  0                0                       0
2                  0                0                       0
3                  0                0                       0
4                  0                0                       1
```

[32]:
```
# Define Predictor and Outcome
X = df_copy.drop('total_benefits',axis=1)
y = df_copy['total_benefits']
```

[33]:
```
# Split the Data - 75% train, 25% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪25,random_state=12345)
```

[34]:
```
# Scaling
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
```

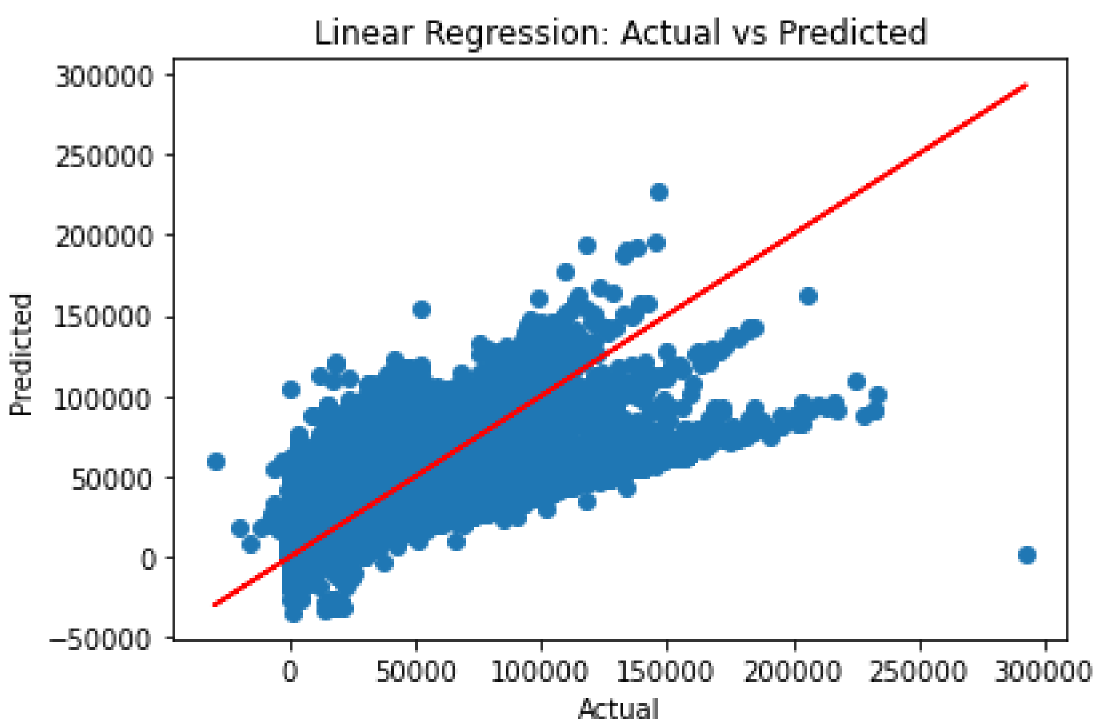
[35]:
```
# Linear Regression
lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled,y_train)
# Prediction
y_pred_lin = lin_reg.predict(X_test_scaled)
R2_lin = metrics.r2_score(y_test, y_pred_lin).round(4)
mae_lin = metrics.mean_absolute_error(y_test, y_pred_lin).round(4)
mse_lin = metrics.mean_squared_error(y_test, y_pred_lin).round(4)
```

```
rmse_lin = np.sqrt(mse_lin).round(4)
# Printing the metrics
# print('Linear Regression goodness of fit: ', lin_reg.score(X_test_scaled,␣
↪y_test).round(4))
print('R2 square:', R2_lin)
print('MAE: ', mae_lin)
print('MSE: ', mse_lin)
print('RMSE: ', rmse_lin)
```

```
Linear Regression Accuracy:  0.7817
R2 square: 0.7817
MAE:  7720.4161
MSE:  127212209.5952
RMSE:  11278.839
```

[36]:
```
plt.scatter(y_test,y_pred_lin)
plt.plot(y_test,y_test, color='red')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Linear Regression: Actual vs Predicted')
```

[36]:
```
Text(0.5, 1.0, 'Linear Regression: Actual vs Predicted')
```

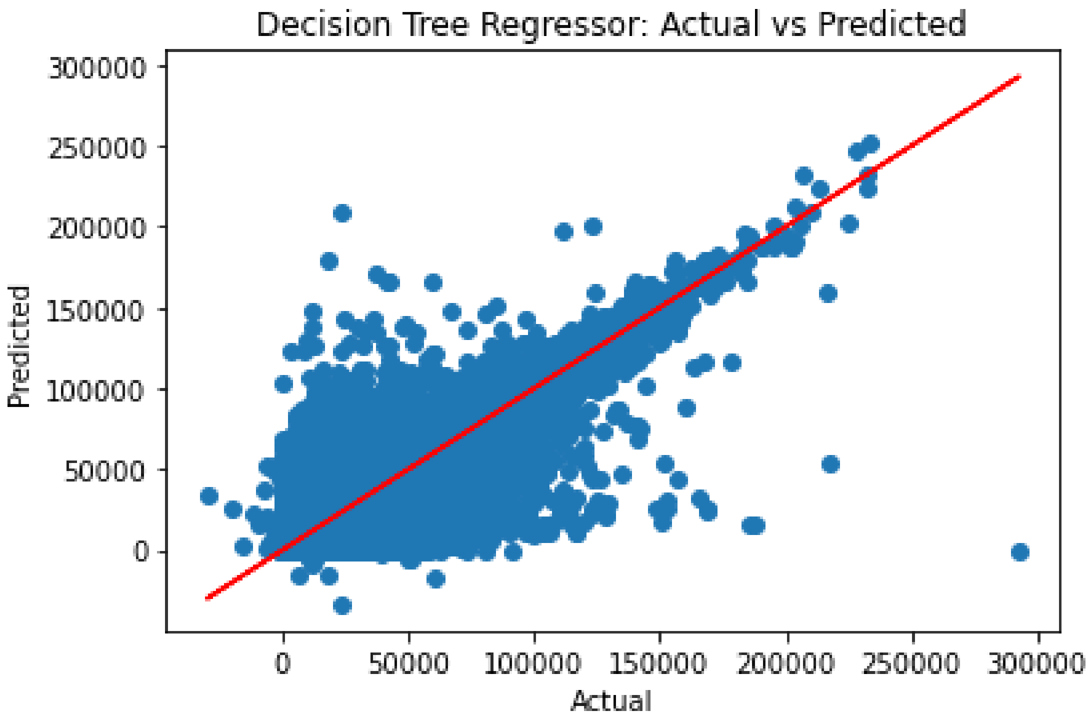


[37]:
```
# Decision Tree
dt_regressor = DecisionTreeRegressor(random_state = 12345)
dt_regressor.fit(X_train_scaled, y_train)
# Prediction
y_pred_dt = dt_regressor.predict(X_test_scaled)
R2_dt = metrics.r2_score(y_test, y_pred_dt).round(4)
mae_dt = metrics.mean_absolute_error(y_test, y_pred_dt).round(4)
mse_dt = metrics.mean_squared_error(y_test, y_pred_dt).round(4)
rmse_dt = np.sqrt(mse_dt).round(4)
# Printing the metrics
# print('Decision Tree Regression goodness of fit: ', dt_regressor.
 ↪score(X_test_scaled,y_test).round(4))
print('R2 square:', R2_dt)
print('MAE: ', mae_dt)
print('MSE: ', mse_dt)
print('RMSE: ', rmse_dt)
```

```
Decision Tree Regression Accuracy:  0.8814
R2 square: 0.8814
MAE:  4271.9505
MSE:  69110903.0235
RMSE:  8313.2968
```

[38]:
```
plt.scatter(y_test,y_pred_dt)
plt.plot(y_test,y_test, color='red')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Decision Tree Regressor: Actual vs Predicted')
```

[38]: Text(0.5, 1.0, 'Decision Tree Regressor: Actual vs Predicted')

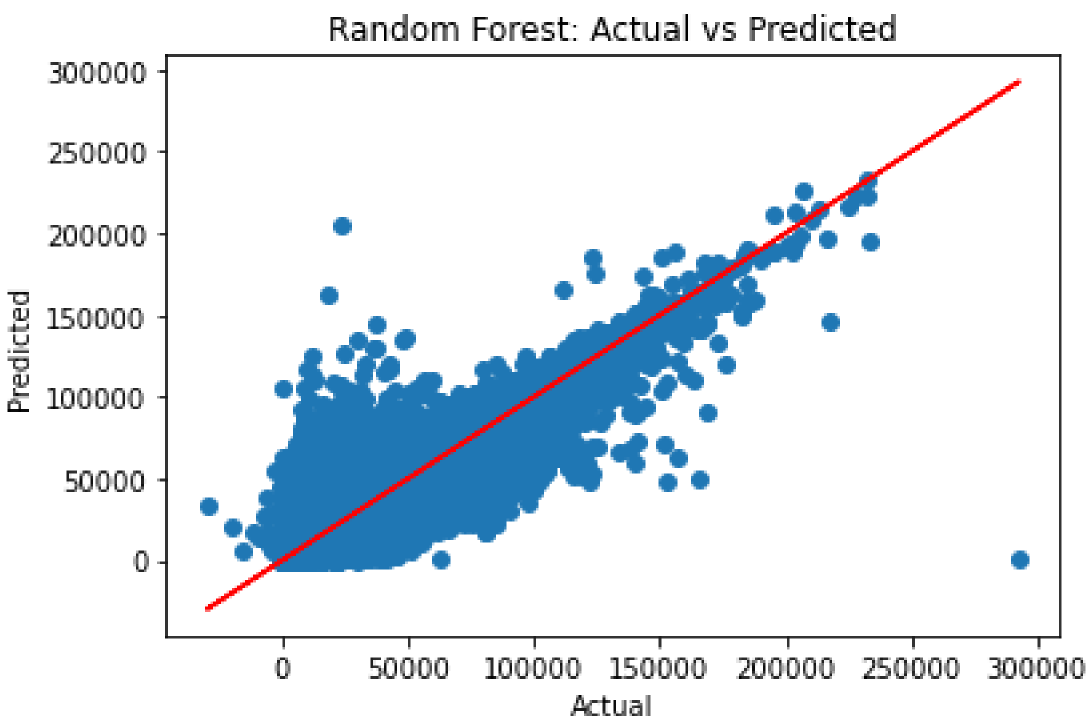


[87]:
```python
# Random Forest Regression
rf_regressor = RandomForestRegressor(n_estimators = 300 , random_state = 12345)
rf_regressor.fit(X_train_scaled, y_train)
# Prediction
y_pred_rf = rf_regressor.predict(X_test_scaled)
R2_rf = metrics.r2_score(y_test, y_pred_rf).round(4)
mae_rf = metrics.mean_absolute_error(y_test, y_pred_rf).round(4)
mse_rf = metrics.mean_squared_error(y_test, y_pred_rf).round(4)
rmse_rf = np.sqrt(mse_rf).round(4)
# Printing the metrics
# print('Random Forest Regression goodness of fit: ', rf_regressor.
 ↪score(X_test_scaled,y_test).round(4))
print('R2 square:', R2_rf)
print('MAE: ', mae_rf)
print('MSE: ', mse_rf)
print('RMSE: ', rmse_rf)
```

```
Random Forest Regression Accuracy:  0.935
R2 square: 0.935
MAE:  3410.7898
MSE:  37859078.3365
RMSE:  6152.9731
```

[88]:
```
plt.scatter(y_test,y_pred_rf)
plt.plot(y_test,y_test, color='red')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Random Forest: Actual vs Predicted')
```

[88]: Text(0.5, 1.0, 'Random Forest: Actual vs Predicted')



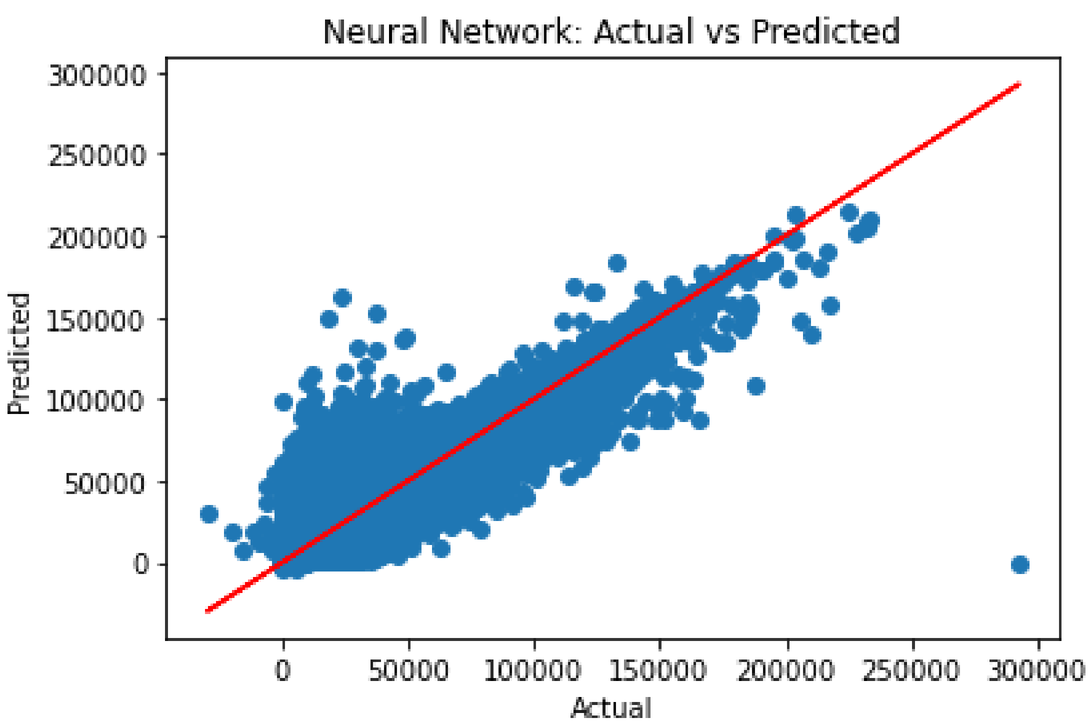
[89]:
```
# Neural Network
mlp_reg = MLPRegressor(hidden_layer_sizes =(150,100,50), max_iter =␣
↪300,activation = 'relu',
solver = 'adam', random_state = 12345)
mlp_reg.fit(X_train_scaled, y_train)
# Prediction
y_pred_nn = mlp_reg.predict(X_test_scaled)
R2_nn = metrics.r2_score(y_test, y_pred_nn).round(4)
mae_nn = metrics.mean_absolute_error(y_test, y_pred_nn).round(4)
mse_nn = metrics.mean_squared_error(y_test, y_pred_nn).round(4)
rmse_nn = np.sqrt(mse_nn).round(4)
# Printing the metrics
# print('Neural Network Regression goodness of fit: ', mlp_reg.
↪score(X_test_scaled,y_test).round(4))
print('R2 square:', R2_nn)
```

```
print('MAE: ', mae_nn)
print('MSE: ', mse_nn)
print('RMSE: ', rmse_nn)
```

```
Neural Network Regression Accuracy:  0.93
R2 square: 0.93
MAE:  3712.015
MSE:  40811691.0904
RMSE:  6388.4029
```

[90]:
```
plt.scatter(y_test,y_pred_nn)
plt.plot(y_test,y_test, color='red')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Neural Network: Actual vs Predicted')
```

[90]:
```
Text(0.5, 1.0, 'Neural Network: Actual vs Predicted')
```



eda_sfr

April 17, 2023

0.1 Linear Learner estimator

[2]:
```
import gzip
import pandas as pd
```

[3]:
```
import numpy as np

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

[4]:
```
with gzip.open('compensation_cpi.csv.gz', 'rb') as f:
    df = pd.read_csv(f)
```

[5]:
```
df.info()
```

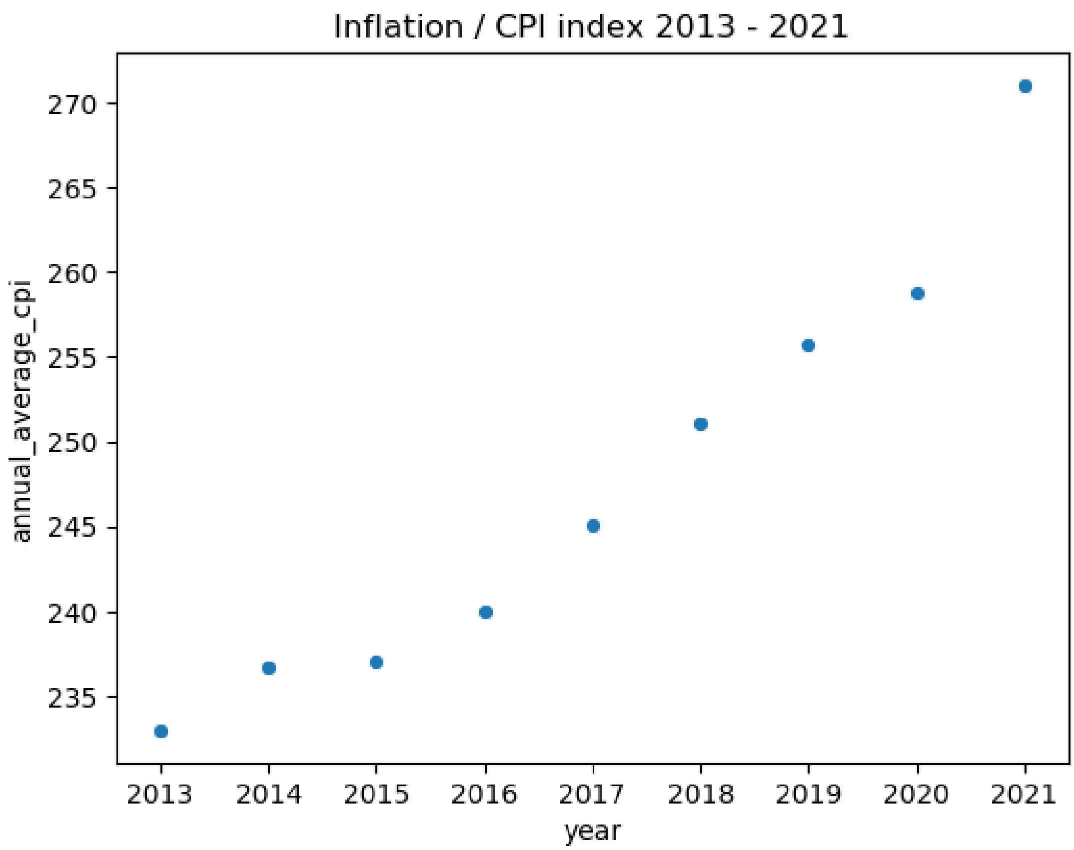
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1466589 entries, 0 to 1466588
Data columns (total 15 columns):
 #   Column               Non-Null Count    Dtype
---  ------               --------------    -----
 0   year                 1466589 non-null  int64
 1   department           1466587 non-null  object
 2   job_title            1466043 non-null  object
 3   base_salary          1465890 non-null  float64
 4   overtime             1433207 non-null  float64
 5   irregular_cash       1466155 non-null  float64
 6   total_cash           1466589 non-null  float64
 7   retirement           1466589 non-null  float64
 8   health               1447602 non-null  float64
 9   other_benefits       830557 non-null   float64
 10  total_benefits       1466589 non-null  float64
 11  total_compensation   1466589 non-null  float64
 12  city_id              1466589 non-null  int64
 13  annual_average_cpi   1466589 non-null  float64
 14  inflation_rate       1466589 non-null  float64
dtypes: float64(11), int64(2), object(2)
memory usage: 167.8+ MB
```

[6]:
```
df.head(3)
```

[6]:
```
   year                  department       job_title  base_salary  \
0  2020  Recreation And Park Commission  Camp Assistant      5257.50
1  2020                       Registrar    Junior Clerk      7699.19
2  2020                       Registrar    Junior Clerk      2619.15

   overtime  irregular_cash  total_cash  retirement  health  other_benefits  \
0       0.0          139.32     5396.82         0.0     0.0          418.88
1    1916.9            0.00     9616.09         0.0     0.0          746.36
2     930.5            0.00     3549.65         0.0     0.0          275.51

   total_benefits  total_compensation  city_id  annual_average_cpi  \
0          418.88             5815.70        2               258.8
1          746.36            10362.45        2               258.8
2          275.51             3825.16        2               258.8

   inflation_rate
0             1.2
1             1.2
2             1.2
```

After pulling in the database, let's view how the CPI has changed over time

[7]:
```
sns.scatterplot(data=df,x='year',y="annual_average_cpi")
plt.title("Inflation / CPI index 2013 - 2021");
```



0.1.1 Condensing Department Names

There are more than 500 unique departments–we want that to be much smaller

[8]:
```
df['department'].value_counts()
```

[8]:
```
POLICE                           136819
WATER AND POWER                  109906
Public Health                     95725
RECREATION AND PARKS              84680
DPH Public Health                 73093
                                  …
Airport-Custodians                    1
Police-Crisis Management              1
Police-TABS                           1
Attorney-Part Time                    1
DOT/Pavement Maint Southeast          1
Name: department, Length: 550, dtype: int64
```

[9]:
```python
#we have two missing department names but they are both police
df[df['department'].isna()]
```

[9]:
```
        year department         job_title  base_salary  overtime  \
139761  2017        NaN   Sheriff's Cadet     49630.50  15016.51
430579  2017        NaN  Police Officer 2    116189.62  40990.09

        irregular_cash  total_cash  retirement    health  other_benefits  \
139761         3197.52    67844.53    10619.27  12779.88         4796.56
430579         2260.08   159439.80    20076.66  14515.01         2724.05

        total_benefits  total_compensation  city_id  annual_average_cpi  \
139761        28195.71            96040.24        2               245.1
430579        37315.72           196755.52        2               245.1

        inflation_rate
139761             2.1
430579             2.1
```

[10]:
```python
#fill in the missing police department names
df['department'].fillna('Police',inplace=True)
```

This function will condense the departments based on a dictionary

[11]:
```python
def replace_text(text):
    if pd.isna(text) or text is None:
        return text
    elif target_word.lower() in text.lower():
        return new_word
    else:
        return text
```

[12]:
```python
#this is the core for how the above functions works
#target_word= "Police"
#new_word= "Police"
#df['department'] = df['department'].apply(replace_text)
```

[13]:
```python
dept_dict = {
    'Police': 'Police', 'Sheriff': 'Police', "Vcet" : "Police",
    "Fire" : "Emergency Management",    "Emergency" : "Emergency Management",
    "PW" : "Public Works",    "Public" : "Public Works",    "Water" : "Public␣
↪Works",    "DOT" : "Public Works",    "Transport" : "Public Works",
    "Plan" : "Public Works",    "Building" : "Public Works",    #"District" :␣
↪"Public Works",
    "PRNS" : "Parks",    "Recre" : "Parks",    "Zoo" : "Parks",    "Parks" :␣
↪"Parks",    "Arena" : "Parks",
```

```
    "City" : "City Mgmt",    "Convention" : "City Mgmt",    "Neighbor" : "City␣
↪Mgmt",    "Election" : "City Mgmt",    "Council" : "City Mgmt",
    "CII" : "City Mgmt",    "Clerk" : "City Mgmt",    "Registrar" : "City␣
↪Mgmt",    "Housing" : "City Mgmt",    "Mayor" : "City Mgmt",    "rda" :␣
↪"City Mgmt",
    "Airport" : "Airport",    "Airside" : "Airport",
    "Finance" : "Finance",    "Auditor" : "Finance",    "Assessor" : "Finance",␣
↪   "Controller" : "Finance",    "Tax" : "Finance", "Treasure" : "Finance",
    "Board" : "Law and Reg",    "Attorney" : "Law and Reg",    "Court" : "Law␣
↪and Reg",
    "Ethics" : "Law and Reg",    "Probation" : "Law and Reg",    "Regulation" :␣
↪"Law and Reg",
    "prt" : "Port",    "port" : "Port", "Harbor" : "Port",
    "Human" : "Human Services",    "Retire" : "Human Services",    "Child" :␣
↪"Human Services",    "Service" : "Human Services",
    "Personnel" : "Human Services",    "Aging" : "Human Services",    "Women" :␣
↪"Human Services",    "Pension" : "Human Services",
    "Disability" : "Human Services",    "Families" : "Human Services", "Youth" :
↪ "Human Services",
    "ESD" : "Human Services",    "Employee" : "Human Services",
    "Info" : "IT",    "Tech" : "IT",
    "Envi" : "Energy, Env, Economy",    "Energy" : "Energy, Env, Economy",   ␣
↪"Power" : "Energy, Env, Economy", "Econ" : "Energy, Env, Economy",
    "Science" : "Libraries, Arts, Science, Museums",    "Librar" : "Libraries,␣
↪Arts, Science, Museums",    "Museum" : "Libraries, Arts, Science, Museums",
    "Memorial" : "Libraries, Arts, Science, Museums",    "Monument" :␣
↪"Libraries, Arts, Science, Museums",    "Arts" : "Libraries, Arts, Science,␣
↪Museums",
    "Cultur" : "Libraries, Arts, Science, Museums", "Art Commission" :␣
↪"Libraries, Arts, Science, Museums"
}
```

[14]:
```
for key in dept_dict:
    target_word= key
    new_word= dept_dict[key]
    df['department'] = df['department'].apply(replace_text)
```

New distribution of departments

[15]:
```
df['department'].value_counts(normalize=True)
```

[15]:
```
Public Works                        0.384807
Police                              0.159051
Port                                0.098913
Parks                               0.097807
Human Services                      0.085051
```

```
Emergency Management                    0.057883
City Mgmt                               0.046575
Libraries, Arts, Science, Museums       0.022667
Law and Reg                             0.019740
Finance                                 0.014328
IT                                      0.008094
Energy, Env, Economy                    0.005083
Name: department, dtype: float64
```

0.1.2 More cleaning

Changing integers for Year and City into categorical variables

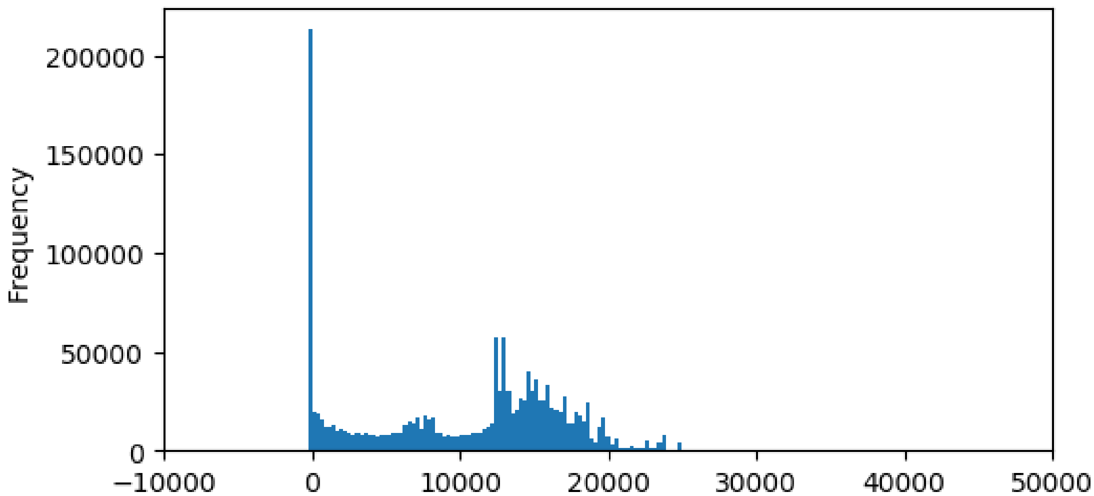
[16]:
```
df['year'] = df['year'].astype('category')
df['city_id'] = df['city_id'].astype('category')
```

[17]:
```
df.isna().sum()
```

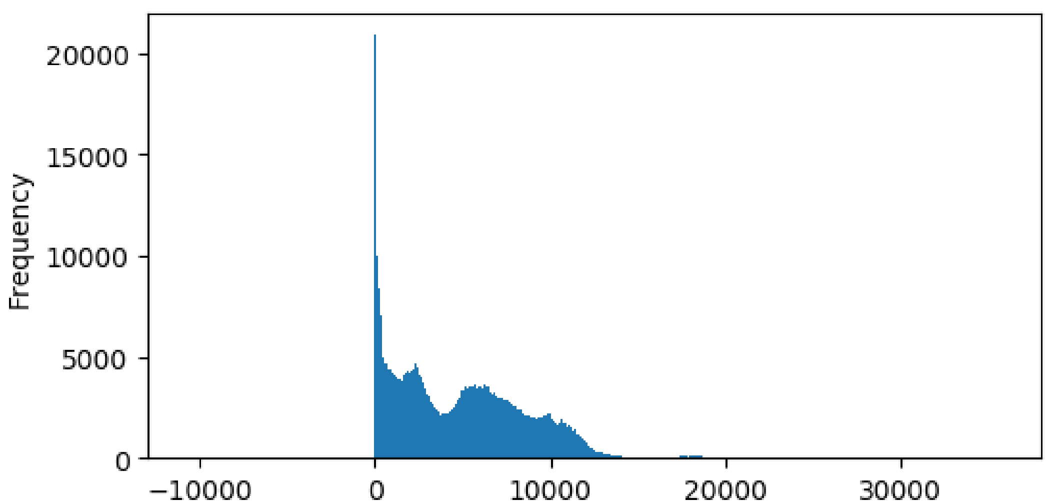
[17]:
```
year                      0
department                0
job_title               546
base_salary             699
overtime              33382
irregular_cash          434
total_cash                0
retirement                0
health                18987
other_benefits       636032
total_benefits            0
total_compensation        0
city_id                   0
annual_average_cpi        0
inflation_rate            0
dtype: int64
```

I feel justified filling in the null values for `health` / `other_benefits` / `overtime` with 0 because it's the single most common practice

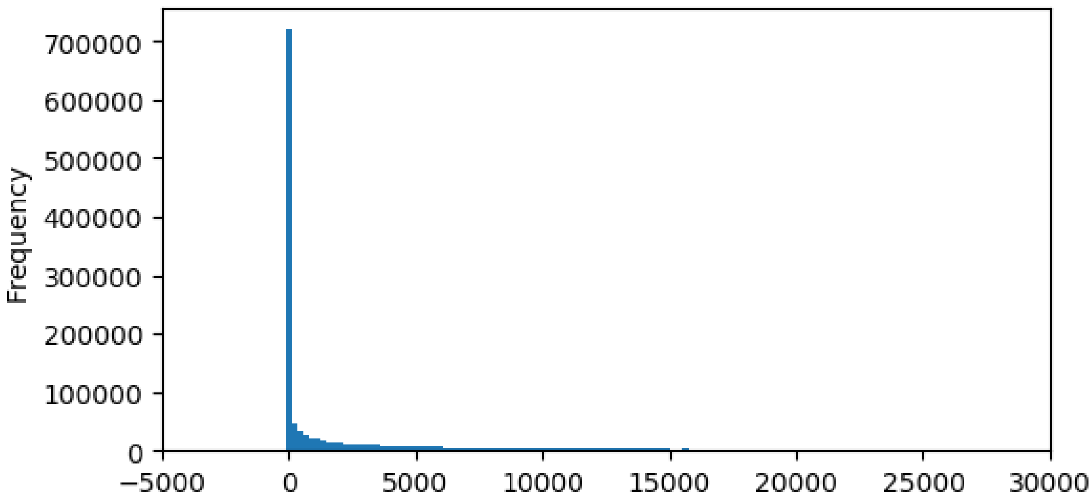
[18]:
```
plt.figure(figsize=(6,3))
plt.xlim(-10000,50000)
df['health'].plot.hist(bins=1000);
```



[19]:
```
plt.figure(figsize=(6,3))
#plt.xlim(-5000,30000)
df['other_benefits'].plot.hist(bins=1000);
```



[20]:
```
plt.figure(figsize=(6,3))
plt.xlim(-5000,30000)
df['overtime'].plot.hist(bins=2000);
```



[21]:
```
df['job_title'] = df['job_title'].fillna("Not disclosed")
df['overtime'] = df['overtime'].fillna(0)
df['irregular_cash'] = df['irregular_cash'].fillna(0)
df['health'] = df['health'].fillna(0)
df['other_benefits'] = df['other_benefits'].fillna(0)

df['base_salary'] = df['base_salary'].fillna((df['total_cash'] - df['overtime']␣
↪- df['irregular_cash']))

df.isna().sum()
```

[21]:
```
year                  0
department            0
job_title             0
base_salary           0
overtime              0
irregular_cash        0
total_cash            0
retirement            0
health                0
other_benefits        0
total_benefits        0
total_compensation    0
city_id               0
annual_average_cpi    0
inflation_rate        0
dtype: int64
```

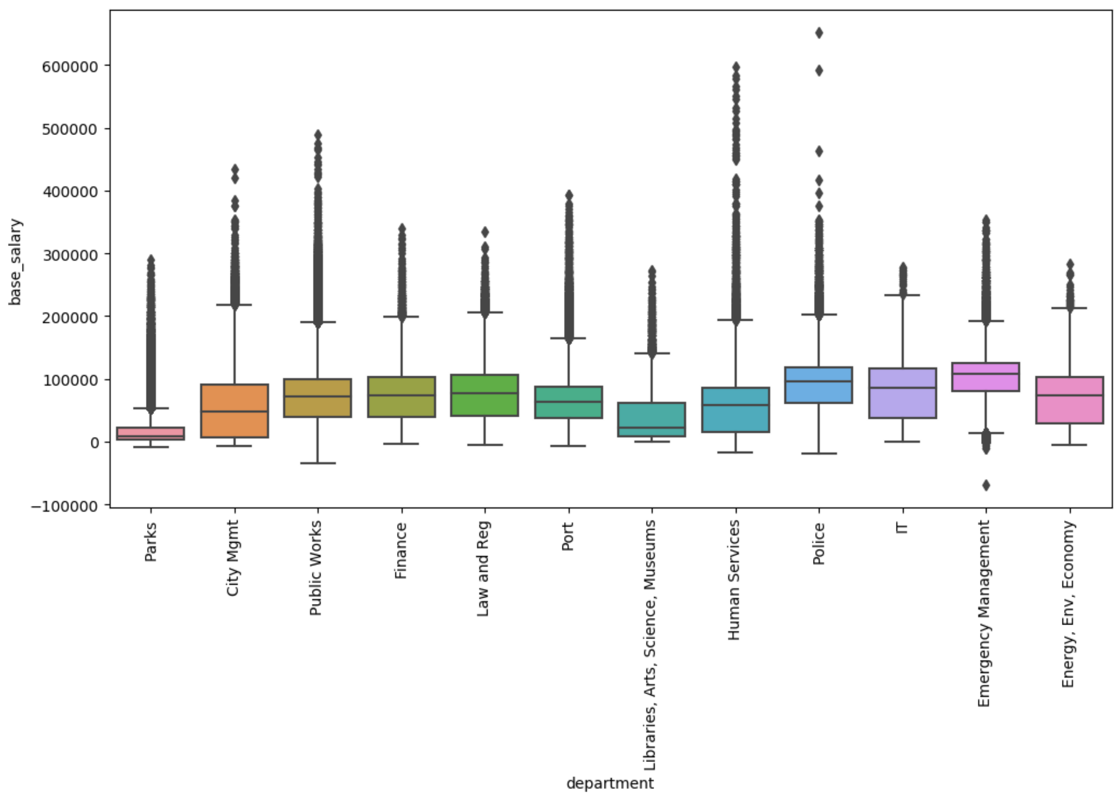
[22]:
```
df[df['base_salary'] < 0]
```

[22]:
```
           year    department               job_title  base_salary  overtime  \
13435      2019          Port        SECURITY OFFICER     -7637.56      0.00
16831      2019  Public Works  PS Aide Health Services      -140.53      0.00
22478      2019         Parks           Not disclosed       -61.44    184.32
27136      2019        Police       POLICE OFFICER II    -14952.00    358.80
27549      2019          Port       GARDENER CARETAKER      -789.64      0.00
…          …             …                      …           …         …
1439466    2021        Police       POLICE OFFICER II      -520.90   6675.33
1444156    2019  Public Works       TRAFFIC OFFICER I     -1760.00      0.00
1452282    2014      City Mgmt      EVENT ATTENDANT II      -111.28    166.92
1453404    2014        Police       POLICE OFFICER II      -120.72   2981.78
1466019    2015        Police       DETENTION OFFICER      -232.95    463.96

         irregular_cash  total_cash  retirement  health  other_benefits  \
13435           7637.56        0.00     1184.13  647.20            0.00
16831              0.00     -140.53      -31.80  591.20          -10.91
22478              0.00      122.88        0.00    0.00            0.00
27136          17985.10     3391.90        0.00    0.00            0.00
27549            789.64        0.00        0.00    0.00            0.00
…                   …           …          …      …              …
1439466            0.00     6154.43        0.00    0.00            0.00
1444156         1760.00        0.00      522.02  328.28            0.00
1452282            1.85       57.49        0.00    0.00            0.00
1453404         8446.12    11307.18      -56.56    0.00            0.00
1466019         3907.92     4138.93        0.00    0.00            0.00

         total_benefits  total_compensation city_id  annual_average_cpi  \
13435           1831.33             1831.33       3               255.7
16831            548.49              407.96       2               255.7
22478              0.00              122.88       3               255.7
27136              0.00             3391.90       3               255.7
27549              0.00                0.00       3               255.7
…                   …                  …      …                  …
1439466            0.00             6154.43       3               271.0
1444156          850.30              850.30       3               255.7
1452282            0.00               57.49       3               236.7
1453404          -56.56            11250.62       3               236.7
1466019            0.00             4138.93       3               237.0

         inflation_rate
13435               1.8
16831               1.8
22478               1.8
27136               1.8
27549               1.8
…                   …
1439466             4.7
```

```
1444156             1.8
1452282             1.6
1453404             1.6
1466019             0.1

[403 rows x 15 columns]
```

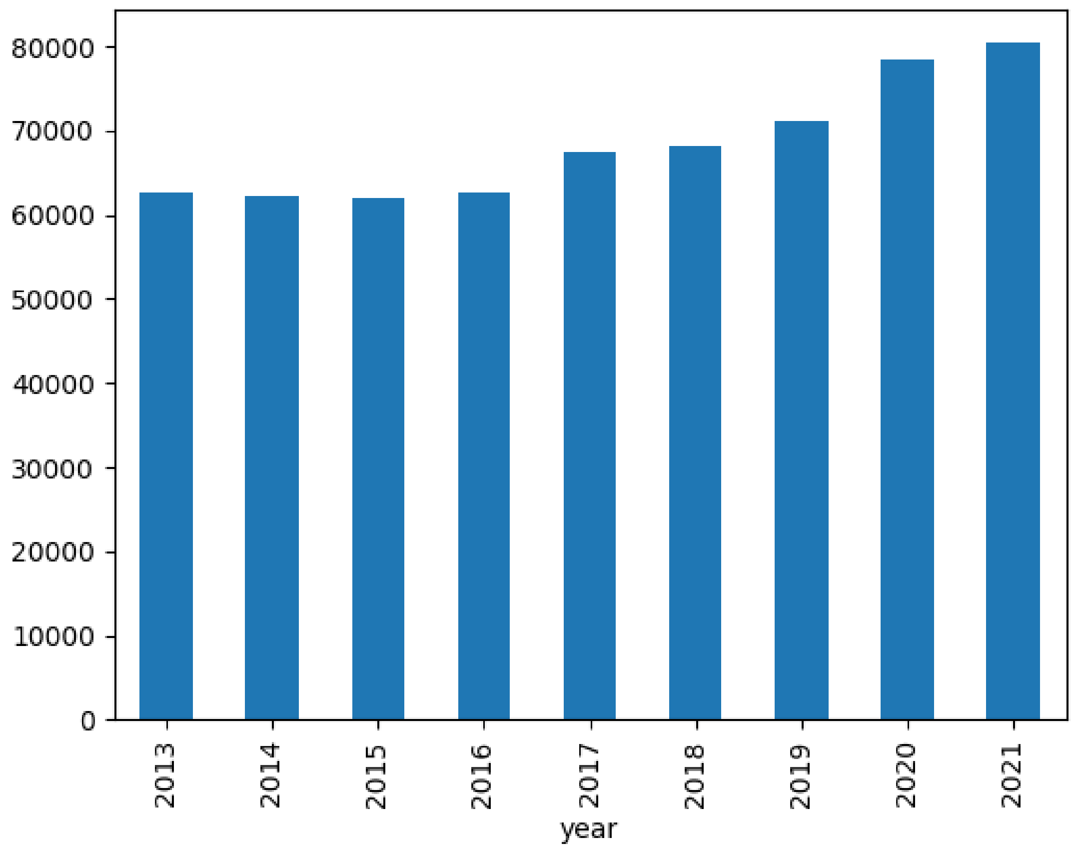
Taking a look at base salary spreads per department

[23]:
```
plt.figure(figsize=(12,6))
sns.boxplot(data=df,x='department',y='base_salary')
plt.xticks(rotation=90);
```



[24]:
```
df.groupby('year')['base_salary'].mean().plot.bar()
```

[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f32b88a7310>



0.1.3 Adjusting all numbers to present-day CPI

[25]:
```
df['base_salary'] = df['base_salary'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['overtime'] = df['overtime'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['irregular_cash'] = df['irregular_cash'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['total_cash'] = df['total_cash'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])

df['retirement'] = df['retirement'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['health'] = df['health'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['other_benefits'] = df['other_benefits'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
df['total_benefits'] = df['total_benefits'] * (df['annual_average_cpi'].max() /␣
 ↪df['annual_average_cpi'])
```

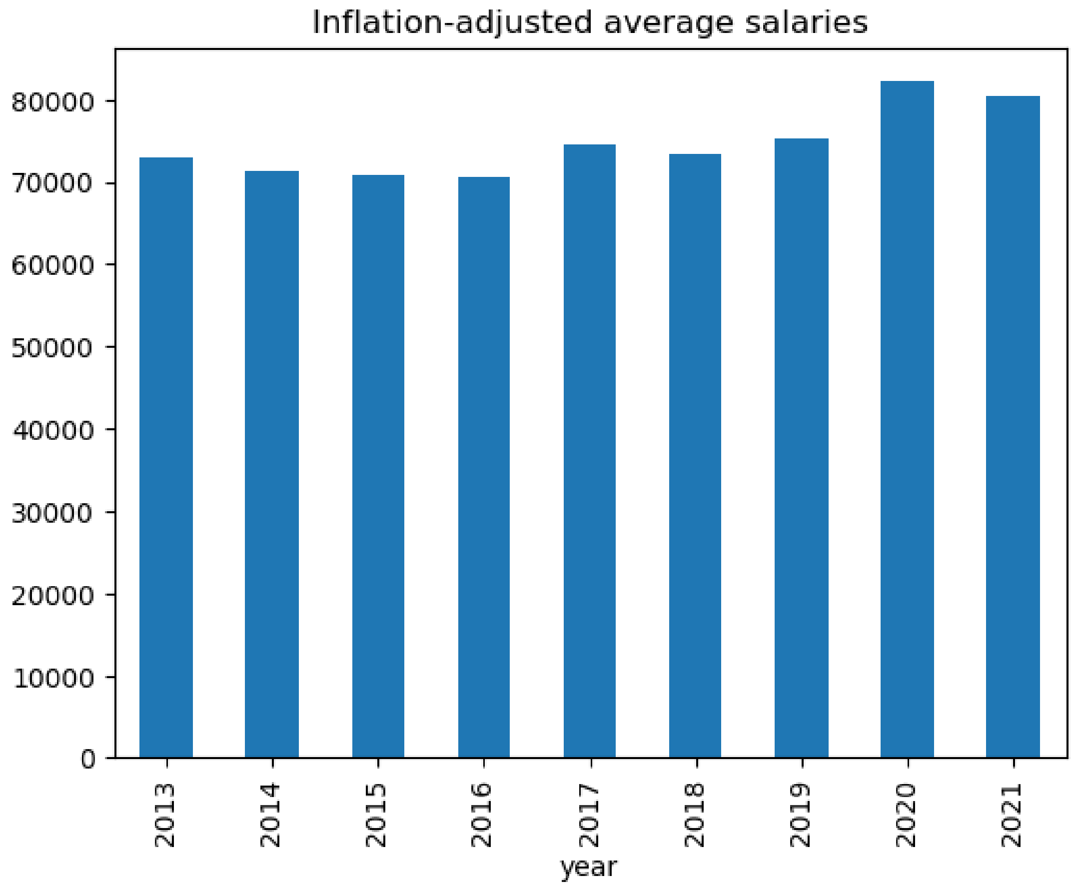
```
df['total_compensation'] = df['total_compensation'] * (df['annual_average_cpi'].
↪max() / df['annual_average_cpi'])
```

[26]:
```
df.head()
```

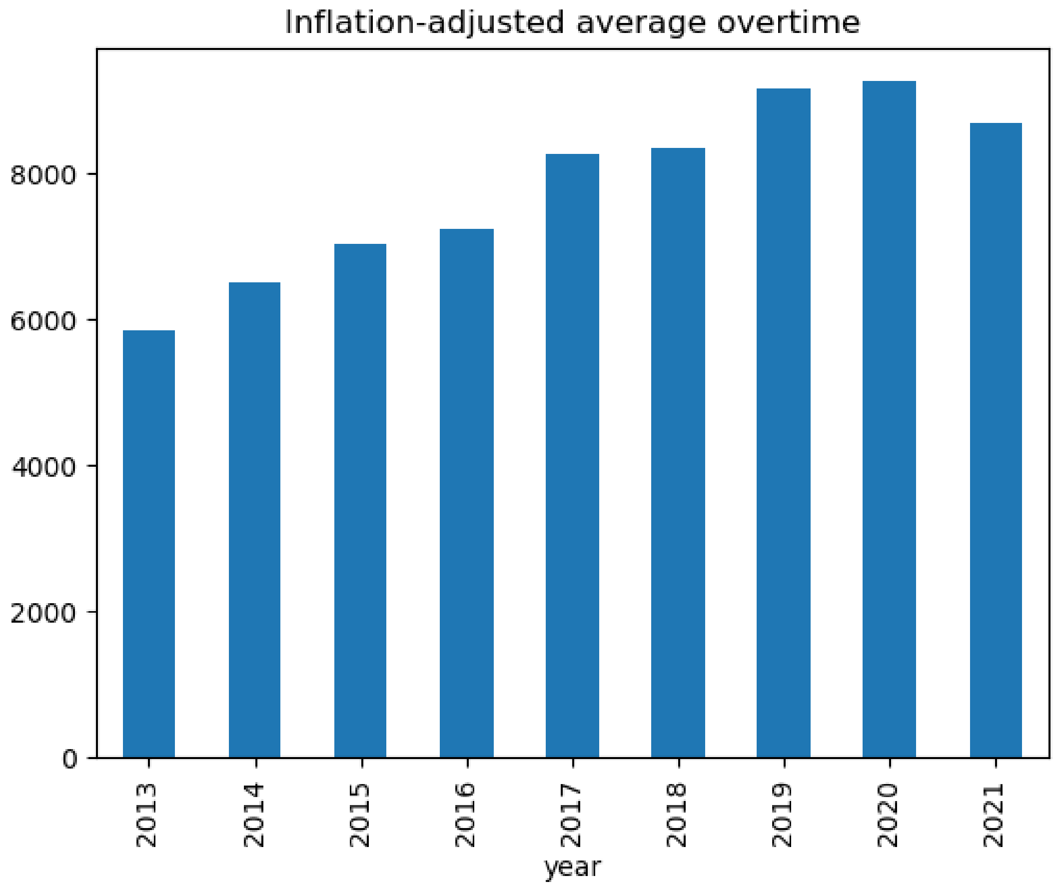
[26]:
```
   year   department        job_title    base_salary     overtime  \
0  2020        Parks   Camp Assistant    5505.341963     0.000000
1  2020    City Mgmt     Junior Clerk    8062.134815  2007.263910
2  2020    City Mgmt     Junior Clerk    2742.618431   974.364374
3  2020    City Mgmt            Clerk    1958.802241   619.090495
4  2020  Public Works        Engineer  166298.647372     0.000000

   irregular_cash     total_cash    retirement        health  other_benefits  \
0      145.887635    5651.229598      0.000000      0.000000      438.626275
1        0.000000   10069.398725      0.000000      0.000000      781.543895
2        0.000000    3716.982805      0.000000      0.000000      288.497720
3        0.000000    2577.892736      0.000000      0.000000      200.077164
4     5944.554637  172243.202009  35106.269861  15799.729328    12219.283192

   total_benefits  total_compensation city_id  annual_average_cpi  \
0      438.626275         6089.855873       2               258.8
1      781.543895        10850.942620       2               258.8
2      288.497720         4005.480526       2               258.8
3      200.077164         2777.969900       2               258.8
4    63125.282380       235368.484389       2               258.8

   inflation_rate
0             1.2
1             1.2
2             1.2
3             1.2
4             1.2
```

0.1.4 Some visuals

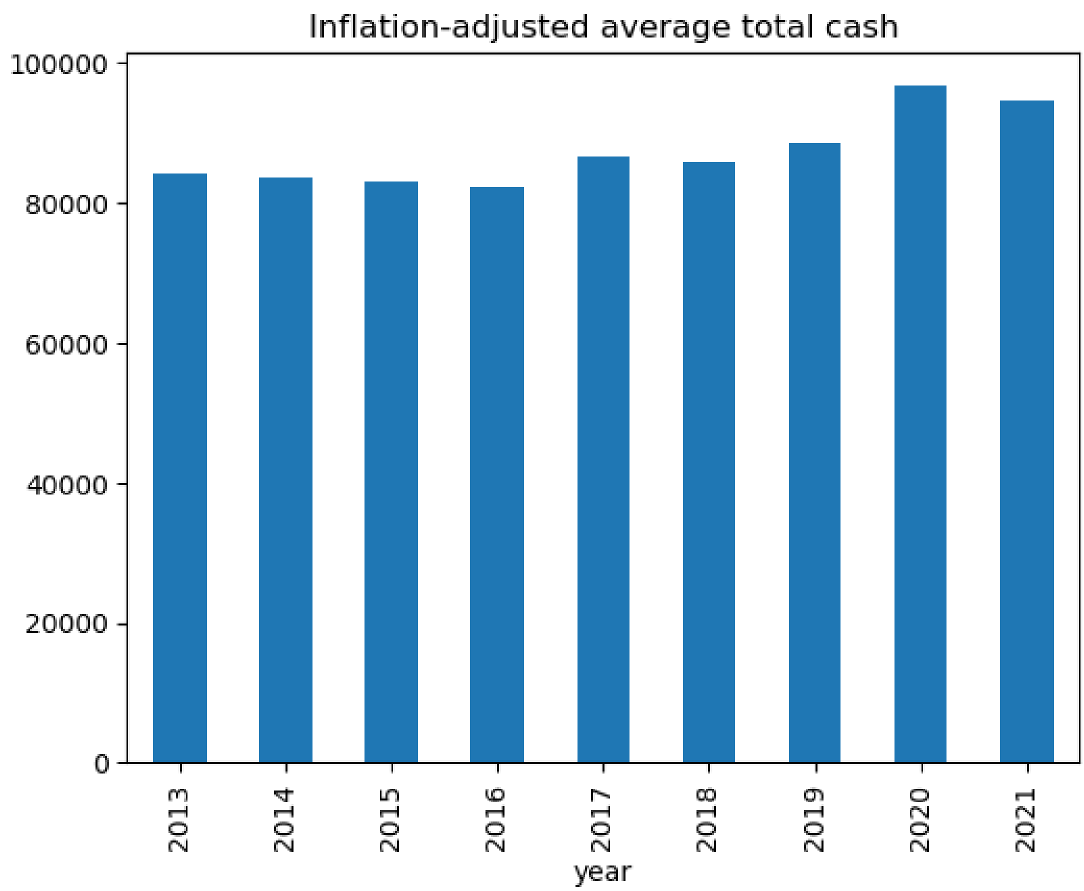
[27]:
```
df.groupby('year')['base_salary'].mean().plot.bar()
plt.title("Inflation-adjusted average salaries");
```



[28]:
```
df.groupby('year')['overtime'].mean().plot.bar()
plt.title("Inflation-adjusted average overtime");
```

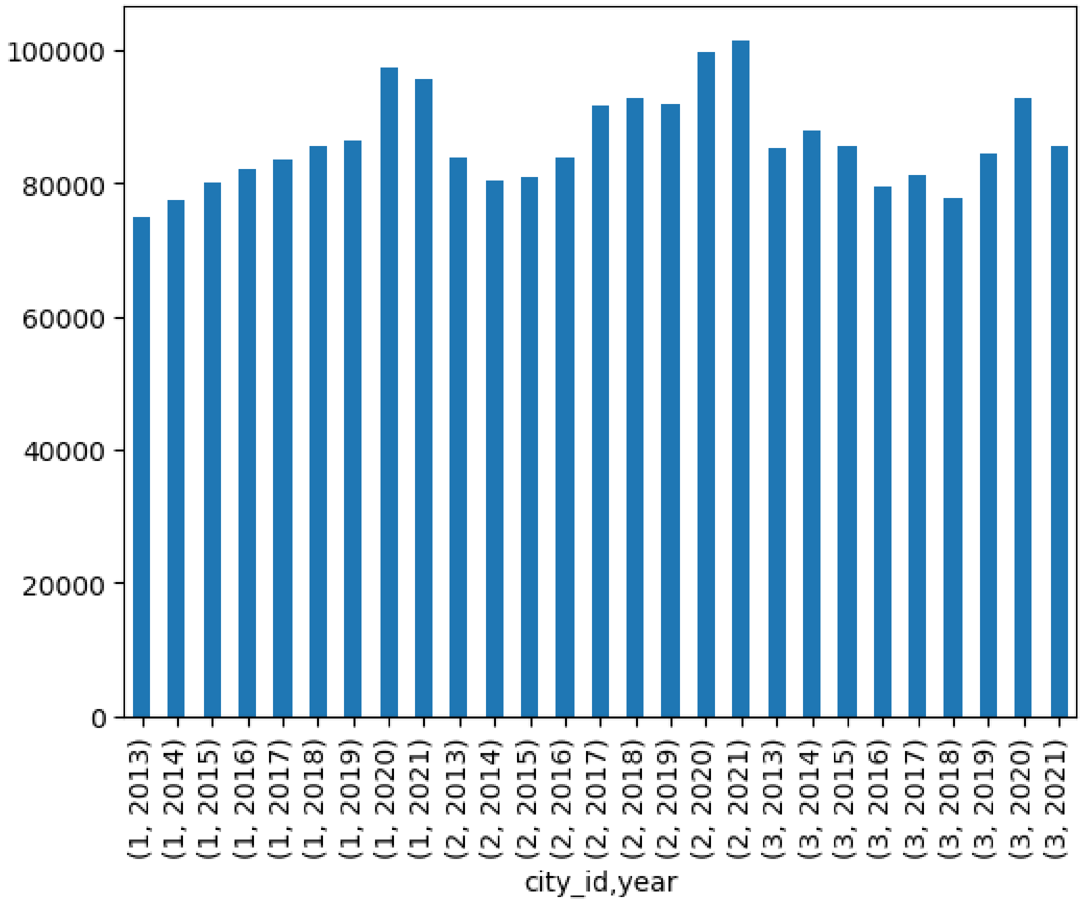


[29]:
```
df.groupby('year')['total_cash'].mean().plot.bar()
plt.title("Inflation-adjusted average total cash");
```




```
[30]: df.groupby(['city_id','year'])['total_cash'].mean().plot.bar()
```

```
[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7f32b11d9e90>
```



[ ]:

```

```

0.2 SageMaker regression

Following along here https://towardsdatascience.com/using-aws-sagemakers-linear-learner-to-solve-regression-problems-36732d802ba6

[31]:

```
import sagemaker
import boto3

sess = sagemaker.Session()
role = sagemaker.get_execution_role()
bucket = sess.default_bucket()
region = boto3.Session().region_name

sm = boto3.Session().client(service_name="sagemaker", region_name=region)
s3 = boto3.Session().client(service_name="s3", region_name=region)
```

[32]:
```
from sagemaker import get_execution_role
from sagemaker.sklearn.processing import SKLearnProcessor

role = get_execution_role()
sklearn_processor = SKLearnProcessor( framework_version="0.20.0", role=role,␣
↪instance_type="ml.m5.xlarge", instance_count=1)
```

[33]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

Converting categorical variables into dummy variables (one-hot encoding)

[35]:
```
pd.get_dummies(df.head(),drop_first=True)
```

[35]:
```
     base_salary     overtime  irregular_cash     total_cash    retirement  \
0    5505.341963     0.000000      145.887635    5651.229598      0.000000
1    8062.134815  2007.263910        0.000000   10069.398725      0.000000
2    2742.618431   974.364374        0.000000    3716.982805      0.000000
3    1958.802241   619.090495        0.000000    2577.892736      0.000000
4  166298.647372     0.000000     5944.554637  172243.202009  35106.269861

         health  other_benefits  total_benefits  total_compensation  \
0      0.000000      438.626275      438.626275         6089.855873
1      0.000000      781.543895      781.543895        10850.942620
2      0.000000      288.497720      288.497720         4005.480526
3      0.000000      200.077164      200.077164         2777.969900
4  15799.729328    12219.283192    63125.282380       235368.484389

   annual_average_cpi  …  year_2019  year_2020  year_2021  department_Parks  \
0               258.8  …          0          1          0                 1
1               258.8  …          0          1          0                 0
2               258.8  …          0          1          0                 0
3               258.8  …          0          1          0                 0
4               258.8  …          0          1          0                 0

   department_Public Works  job_title_Clerk  job_title_Engineer  \
0                        0                0                   0
1                        0                0                   0
2                        0                0                   0
3                        0                1                   0
4                        1                0                   1

   job_title_Junior Clerk  city_id_2  city_id_3
0                       0          1          0
1                       1          1          0
2                       1          1          0
3                       0          1          0
```

```
4                        0           1           0

[5 rows x 26 columns]
```

Setting up X and y train and test data sets in a standard manner

[82]:
```
X = pd.get_dummies(df[['base_salary', 'overtime', 'irregular_cash', 'year', ␣
 ↪'department',  'annual_average_cpi', 'city_id']],drop_first=True)
y = df['total_benefits']
```

[99]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state = 0)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

y_train = y_train.astype("float32")
y_test = y_test.astype("float32")
```

Scaling the data

[100]:
```
sc = StandardScaler()

X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.transform(X_test)
```

Following along with external resource linked above

[101]:
```
prefix = "linear-learner"
```

[102]:
```
import io
import sagemaker.amazon.common as smac
import os
```

[103]:
```
#upload training data
buf = io.BytesIO()
smac.write_numpy_to_dense_tensor(buf, X_train_sc, y_train.
 ↪reset_index(drop=True))
buf.seek(0)

key = 'linear-train-data'

boto3.resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train', key)).
 ↪upload_fileobj(buf)
s3_train_data = 's3://{}/{}/train/{}'.format(bucket, prefix, key)

print('uploaded training data location: {}'.format(s3_train_data))
```

```
uploaded training data location: s3://sagemaker-us-east-1-117315948243/linear-
learner/train/linear-train-data
```

[104]:
```python
buf = io.BytesIO()
smac.write_numpy_to_dense_tensor(buf, X_test_sc, y_test.reset_index(drop=True))
buf.seek(0)

key = 'linear-test-data'

boto3.resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'test', key)).
 ↪upload_fileobj(buf)
s3_test_data = 's3://{}/{}/test/{}'.format(bucket, prefix, key)

print('uploaded training data location: {}'.format(s3_test_data))
```

```
uploaded training data location: s3://sagemaker-us-east-1-117315948243/linear-
learner/test/linear-test-data
```

[105]:
```python
output_location = 's3://{}/{}/output'.format(bucket, prefix)
print('Training artifacts will be uploaded to: {}'.format(output_location))
```

```
Training artifacts will be uploaded to: s3://sagemaker-us-
east-1-117315948243/linear-learner/output
```

Training Linear Learner

[106]:
```python
from sagemaker.amazon.amazon_estimator import image_uris
```

[107]:
```python
container = image_uris.retrieve('linear-learner', boto3.Session().region_name)

linear = sagemaker.estimator.Estimator(container,
                                       role,
                                       instance_count = 1,
                                       instance_type = 'ml.c4.xlarge',
                                       output_path = output_location,
                                       sagemaker_session = sess)
```

```
INFO:sagemaker.image_uris:Same images used for training and inference.
Defaulting to image scope: inference.
INFO:sagemaker.image_uris:Defaulting to the only supported framework/algorithm
version: 1.
INFO:sagemaker.image_uris:Ignoring unnecessary instance type: None.
```

[108]:
```python
linear.set_hyperparameters(feature_dim = 25,
                           predictor_type = 'regressor',
                           mini_batch_size = 20,
                           epochs = 5,
                           num_models = 10,
```

```
                        loss = 'absolute_loss')
```

[109]:
```
linear.fit({'train': s3_train_data})
```

```
INFO:sagemaker:Creating training-job with name: linear-
learner-2023-04-06-21-42-00-708

2023-04-06 21:42:01 Starting - Starting the training job…
2023-04-06 21:42:19 Starting - Preparing the instances for training…
2023-04-06 21:43:27 Downloading - Downloading input data…
2023-04-06 21:43:58 Training - Downloading the training image…Docker
entrypoint called with argument(s): train
Running default environment configuration script
[04/06/2023 21:45:25 INFO 139636117940032] Reading default configuration
from /opt/amazon/lib/python3.7/site-packages/algorithm/resources/default-
input.json: {'mini_batch_size': '1000', 'epochs': '15', 'feature_dim': 'auto',
'use_bias': 'true', 'binary_classifier_model_selection_criteria': 'accuracy',
'f_beta': '1.0', 'target_recall': '0.8', 'target_precision': '0.8',
'num_models': 'auto', 'num_calibration_samples': '10000000', 'init_method':
'uniform', 'init_scale': '0.07', 'init_sigma': '0.01', 'init_bias': '0.0',
'optimizer': 'auto', 'loss': 'auto', 'margin': '1.0', 'quantile': '0.5',
'loss_insensitivity': '0.01', 'huber_delta': '1.0', 'num_classes': '1',
'accuracy_top_k': '3', 'wd': 'auto', 'l1': 'auto', 'momentum': 'auto',
'learning_rate': 'auto', 'beta_1': 'auto', 'beta_2': 'auto', 'bias_lr_mult':
'auto', 'bias_wd_mult': 'auto', 'use_lr_scheduler': 'true', 'lr_scheduler_step':
'auto', 'lr_scheduler_factor': 'auto', 'lr_scheduler_minimum_lr': 'auto',
'positive_example_weight_mult': '1.0', 'balance_multiclass_weights': 'false',
'normalize_data': 'true', 'normalize_label': 'auto', 'unbias_data': 'auto',
'unbias_label': 'auto', 'num_point_for_scaler': '10000', '_kvstore': 'auto',
'_num_gpus': 'auto', '_num_kv_servers': 'auto', '_log_level': 'info',
'_tuning_objective_metric': '', 'early_stopping_patience': '3',
'early_stopping_tolerance': '0.001', '_enable_profiler': 'false'}
[04/06/2023 21:45:25 INFO 139636117940032] Merging with provided
configuration from /opt/ml/input/config/hyperparameters.json: {'epochs': '5',
'feature_dim': '25', 'loss': 'absolute_loss', 'mini_batch_size': '20',
'num_models': '10', 'predictor_type': 'regressor'}
```

```
[04/06/2023 21:45:25 INFO 139636117940032] Final configuration:
{'mini_batch_size': '20', 'epochs': '5', 'feature_dim': '25', 'use_bias':
'true', 'binary_classifier_model_selection_criteria': 'accuracy', 'f_beta':
'1.0', 'target_recall': '0.8', 'target_precision': '0.8', 'num_models': '10',
'num_calibration_samples': '10000000', 'init_method': 'uniform', 'init_scale':
'0.07', 'init_sigma': '0.01', 'init_bias': '0.0', 'optimizer': 'auto', 'loss':
'absolute_loss', 'margin': '1.0', 'quantile': '0.5', 'loss_insensitivity':
'0.01', 'huber_delta': '1.0', 'num_classes': '1', 'accuracy_top_k': '3', 'wd':
'auto', 'l1': 'auto', 'momentum': 'auto', 'learning_rate': 'auto', 'beta_1':
'auto', 'beta_2': 'auto', 'bias_lr_mult': 'auto', 'bias_wd_mult': 'auto',
'use_lr_scheduler': 'true', 'lr_scheduler_step': 'auto', 'lr_scheduler_factor':
'auto', 'lr_scheduler_minimum_lr': 'auto', 'positive_example_weight_mult':
'1.0', 'balance_multiclass_weights': 'false', 'normalize_data': 'true',
'normalize_label': 'auto', 'unbias_data': 'auto', 'unbias_label': 'auto',
'num_point_for_scaler': '10000', '_kvstore': 'auto', '_num_gpus': 'auto',
'_num_kv_servers': 'auto', '_log_level': 'info', '_tuning_objective_metric': '',
'early_stopping_patience': '3', 'early_stopping_tolerance': '0.001',
'_enable_profiler': 'false', 'predictor_type': 'regressor'}
[04/06/2023 21:45:28 WARNING 139636117940032] Loggers have already been
setup.
```

```
[04/06/2023 21:45:28 INFO 139636117940032] Final configuration:
{'mini_batch_size': '20', 'epochs': '5', 'feature_dim': '25', 'use_bias':
'true', 'binary_classifier_model_selection_criteria': 'accuracy', 'f_beta':
'1.0', 'target_recall': '0.8', 'target_precision': '0.8', 'num_models': '10',
'num_calibration_samples': '10000000', 'init_method': 'uniform', 'init_scale':
'0.07', 'init_sigma': '0.01', 'init_bias': '0.0', 'optimizer': 'auto', 'loss':
'absolute_loss', 'margin': '1.0', 'quantile': '0.5', 'loss_insensitivity':
'0.01', 'huber_delta': '1.0', 'num_classes': '1', 'accuracy_top_k': '3', 'wd':
'auto', 'l1': 'auto', 'momentum': 'auto', 'learning_rate': 'auto', 'beta_1':
'auto', 'beta_2': 'auto', 'bias_lr_mult': 'auto', 'bias_wd_mult': 'auto',
'use_lr_scheduler': 'true', 'lr_scheduler_step': 'auto', 'lr_scheduler_factor':
'auto', 'lr_scheduler_minimum_lr': 'auto', 'positive_example_weight_mult':
'1.0', 'balance_multiclass_weights': 'false', 'normalize_data': 'true',
'normalize_label': 'auto', 'unbias_data': 'auto', 'unbias_label': 'auto',
'num_point_for_scaler': '10000', '_kvstore': 'auto', '_num_gpus': 'auto',
'_num_kv_servers': 'auto', '_log_level': 'info', '_tuning_objective_metric': '',
'early_stopping_patience': '3', 'early_stopping_tolerance': '0.001',
'_enable_profiler': 'false', 'predictor_type': 'regressor'}
[04/06/2023 21:45:28 WARNING 139636117940032] Loggers have already been
setup.
Process 7 is a worker.
[04/06/2023 21:45:28 INFO 139636117940032] Using default worker.
[04/06/2023 21:45:28 INFO 139636117940032] Checkpoint loading and saving
are disabled.
[2023-04-06 21:45:28.695] [tensorio] [info] epoch_stats={"data_pipeline":
"/opt/ml/input/data/train", "epoch": 0, "duration": 34, "num_examples": 1,
"num_bytes": 2880}
[04/06/2023 21:45:28 INFO 139636117940032] Create Store: local

2023-04-06 21:45:18 Training - Training image download completed. Training in
progress.[2023-04-06 21:45:31.996] [tensorio] [info]
epoch_stats={"data_pipeline": "/opt/ml/input/data/train", "epoch": 1,
"duration": 3300, "num_examples": 501, "num_bytes": 1442880}
[04/06/2023 21:45:31 INFO 139636117940032] Scaler algorithm parameters
 <algorithm.scaler.ScalerAlgorithmStable object at 0x7eff009854d0>
```

```
[04/06/2023 21:45:31 INFO 139636117940032] Scaling model computed with
parameters:
 {'stdev_label':
[24151.914]
<NDArray 1 @cpu(0)>, 'stdev_weight':
[1.0025886  0.9798619  1.0517213  0.9833616  0.9938397  1.0055946
 1.0031267  1.0087105  1.0286226  1.0008031  0.98754025 0.96758264
 0.9833613  1.0037589  1.0357529  0.9962312  1.0071249  1.007429
 1.0013174  0.99462456 0.9987299  1.0222116  0.99934596 0.9999126
 0.999704  ]
<NDArray 25 @cpu(0)>, 'mean_label':
[32464.303]
<NDArray 1 @cpu(0)>, 'mean_weight':
[ 1.4473288e-02  3.4424127e-05  1.2415959e-02 -1.6292194e-02
 -4.7600796e-03  4.5174169e-03  2.6007164e-03  7.1570054e-03
  2.4697648e-02  6.7498034e-04 -1.0131971e-02 -2.4697363e-02
 -8.7080505e-03  5.3957407e-04  8.9089014e-03 -2.5263566e-03
  1.3082669e-03  2.1557948e-03  4.1018651e-04 -3.9560972e-03
 -1.3596788e-03  1.6840592e-02 -2.7397729e-03  2.4492587e-03
 -2.1974726e-03]
<NDArray 25 @cpu(0)>}
[04/06/2023 21:45:32 INFO 139636117940032] nvidia-smi: took 0.032 seconds
to run.
[04/06/2023 21:45:32 INFO 139636117940032] nvidia-smi identified 0
GPUs.
[04/06/2023 21:45:32 INFO 139636117940032] Number of GPUs being used: 0
```
#metrics {"StartTime": 1680817532.0524197, "EndTime": 1680817532.0524564,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "Meta": "init_train_data_iter"}, "Metrics": {"Total Records Seen":
{"sum": 10040.0, "count": 1, "min": 10040, "max": 10040}, "Total Batches Seen":
{"sum": 502.0, "count": 1, "min": 502, "max": 502}, "Max Records Seen Between
Resets": {"sum": 10020.0, "count": 1, "min": 10020, "max": 10020}, "Max Batches
Seen Between Resets": {"sum": 501.0, "count": 1, "min": 501, "max": 501}, "Reset
Count": {"sum": 2.0, "count": 1, "min": 2, "max": 2}, "Number of Records Since
Last Reset": {"sum": 0.0, "count": 1, "min": 0, "max": 0}, "Number of Batches
Since Last Reset": {"sum": 0.0, "count": 1, "min": 0, "max": 0}}}

[2023-04-06 21:51:19.835] [tensorio] [info] epoch_stats={"data_pipeline": "/opt/ml/input/data/train", "epoch": 3, "duration": 347782, "num_examples": 51331, "num_bytes": 147832128}
#metrics {"StartTime": 1680817879.835465, "EndTime": 1680817879.8355315, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 0, "model": 0}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3073006487771732, "count": 1, "min": 0.3073006487771732, "max": 0.3073006487771732}}}
#metrics {"StartTime": 1680817879.835672, "EndTime": 1680817879.8356924, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 0, "model": 1}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3090235244361555, "count": 1, "min": 0.3090235244361555, "max": 0.3090235244361555}}}
#metrics {"StartTime": 1680817879.8357458, "EndTime": 1680817879.8357577, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 0, "model": 2}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30742185259617816, "count": 1, "min": 0.30742185259617816, "max": 0.30742185259617816}}}
#metrics {"StartTime": 1680817879.835801, "EndTime": 1680817879.8358116, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 0, "model": 3}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30900738624390306, "count": 1, "min": 0.30900738624390306, "max": 0.30900738624390306}}}
#metrics {"StartTime": 1680817879.8358524, "EndTime": 1680817879.8358622, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 0, "model": 4}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.31050350865216, "count": 1, "min": 0.31050350865216, "max": 0.31050350865216}}}
#metrics {"StartTime": 1680817879.8359017, "EndTime": 1680817879.835912, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 0, "model": 5}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3467534000041446, "count": 1, "min": 0.3467534000041446, "max": 0.3467534000041446}}}

```
#metrics {"StartTime": 1680817879.835951, "EndTime": 1680817879.835961,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 0, "model": 6}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.3108230959375119, "count": 1, "min":
0.3108230959375119, "max": 0.3108230959375119}}}
#metrics {"StartTime": 1680817879.836, "EndTime": 1680817879.83601,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 0, "model": 7}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.3473019189987177, "count": 1, "min":
0.3473019189987177, "max": 0.3473019189987177}}}
#metrics {"StartTime": 1680817879.8360488, "EndTime": 1680817879.8360589,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 0, "model": 8}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.3072909878752001, "count": 1, "min":
0.3072909878752001, "max": 0.3072909878752001}}}
#metrics {"StartTime": 1680817879.836098, "EndTime": 1680817879.836108,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 0, "model": 9}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.3091402463717417, "count": 1, "min":
0.3091402463717417, "max": 0.3091402463717417}}}
[04/06/2023 21:51:19 INFO 139636117940032] #quality_metric: host=algo-1,
epoch=0, train absolute_loss_objective <loss>=0.3073006487771732
[04/06/2023 21:51:19 INFO 139636117940032] #early_stopping_criteria_metric:
host=algo-1, epoch=0, criteria=absolute_loss_objective,
value=0.3072909878752001
[04/06/2023 21:51:19 INFO 139636117940032] Epoch 0: Loss improved. Updating
best model
[04/06/2023 21:51:19 INFO 139636117940032] Saving model for epoch: 0
[04/06/2023 21:51:19 INFO 139636117940032] Saved checkpoint to
"/tmp/tmprga7r20e/mx-mod-0000.params"
[04/06/2023 21:51:19 INFO 139636117940032] #progress_metric: host=algo-1,
completed 20.0 % of epochs
```

#metrics {"StartTime": 1680817532.0527554, "EndTime": 1680817879.8455803, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 0, "Meta": "training_data_iter"}, "Metrics": {"Total Records Seen": {"sum": 1036652.0, "count": 1, "min": 1036652, "max": 1036652}, "Total Batches Seen": {"sum": 51833.0, "count": 1, "min": 51833, "max": 51833}, "Max Records Seen Between Resets": {"sum": 1026612.0, "count": 1, "min": 1026612, "max": 1026612}, "Max Batches Seen Between Resets": {"sum": 51331.0, "count": 1, "min": 51331, "max": 51331}, "Reset Count": {"sum": 3.0, "count": 1, "min": 3, "max": 3}, "Number of Records Since Last Reset": {"sum": 1026612.0, "count": 1, "min": 1026612, "max": 1026612}, "Number of Batches Since Last Reset": {"sum": 51331.0, "count": 1, "min": 51331, "max": 51331}}}
[04/06/2023 21:51:19 INFO 139636117940032] #throughput_metric: host=algo-1, train throughput=2951.7904915476743 records/second
[2023-04-06 21:56:59.923] [tensorio] [info] epoch_stats={"data_pipeline": "/opt/ml/input/data/train", "epoch": 5, "duration": 340076, "num_examples": 51331, "num_bytes": 147832128}
#metrics {"StartTime": 1680818219.9233587, "EndTime": 1680818219.9234238, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 1, "model": 0}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30634301322421065, "count": 1, "min": 0.30634301322421065, "max": 0.30634301322421065}}}
#metrics {"StartTime": 1680818219.9235244, "EndTime": 1680818219.923547, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 1, "model": 1}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.306350212125153, "count": 1, "min": 0.306350212125153, "max": 0.306350212125153}}}
#metrics {"StartTime": 1680818219.9236164, "EndTime": 1680818219.9236352, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 1, "model": 2}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3065107272296256, "count": 1, "min": 0.3065107272296256, "max": 0.3065107272296256}}}

#metrics {"StartTime": 1680818219.9236944, "EndTime": 1680818219.923712, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 1, "model": 3}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3065102827942629, "count": 1, "min": 0.3065102827942629, "max": 0.3065102827942629}}}
#metrics {"StartTime": 1680818219.9237683, "EndTime": 1680818219.9237847, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 1, "model": 4}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.306343804805478, "count": 1, "min": 0.306343804805478, "max": 0.306343804805478}}}
#metrics {"StartTime": 1680818219.9238513, "EndTime": 1680818219.9238684, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 1, "model": 5}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3065510535215788, "count": 1, "min": 0.3065510535215788, "max": 0.3065510535215788}}}
#metrics {"StartTime": 1680818219.9239314, "EndTime": 1680818219.9239485, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 1, "model": 6}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30651117248317317, "count": 1, "min": 0.30651117248317317, "max": 0.30651117248317317}}}
#metrics {"StartTime": 1680818219.9240043, "EndTime": 1680818219.9240208, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 1, "model": 7}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3066280195932743, "count": 1, "min": 0.3066280195932743, "max": 0.3066280195932743}}}
#metrics {"StartTime": 1680818219.9240832, "EndTime": 1680818219.9241002, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 1, "model": 8}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3063595339340609, "count": 1, "min": 0.3063595339340609, "max": 0.3063595339340609}}}

```
#metrics {"StartTime": 1680818219.9241629, "EndTime": 1680818219.9241807, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 1, "model": 9}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30637006903180086, "count": 1, "min": 0.30637006903180086, "max": 0.30637006903180086}}}
[04/06/2023 21:56:59 INFO 139636117940032] #quality_metric: host=algo-1, epoch=1, train absolute_loss_objective <loss>=0.30634301322421065
[04/06/2023 21:56:59 INFO 139636117940032] #early_stopping_criteria_metric: host=algo-1, epoch=1, criteria=absolute_loss_objective, value=0.30634301322421065
[04/06/2023 21:56:59 INFO 139636117940032] Epoch 1: Loss improved. Updating best model
[04/06/2023 21:56:59 INFO 139636117940032] Saving model for epoch: 1
[04/06/2023 21:56:59 INFO 139636117940032] Saved checkpoint to "/tmp/tmplf9k3xa5/mx-mod-0000.params"
[04/06/2023 21:56:59 INFO 139636117940032] #progress_metric: host=algo-1, completed 40.0 % of epochs
#metrics {"StartTime": 1680817879.8467882, "EndTime": 1680818219.9313674, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 1, "Meta": "training_data_iter"}, "Metrics": {"Total Records Seen": {"sum": 2063264.0, "count": 1, "min": 2063264, "max": 2063264}, "Total Batches Seen": {"sum": 103164.0, "count": 1, "min": 103164, "max": 103164}, "Max Records Seen Between Resets": {"sum": 1026612.0, "count": 1, "min": 1026612, "max": 1026612}, "Max Batches Seen Between Resets": {"sum": 51331.0, "count": 1, "min": 51331, "max": 51331}, "Reset Count": {"sum": 4.0, "count": 1, "min": 4, "max": 4}, "Number of Records Since Last Reset": {"sum": 1026612.0, "count": 1, "min": 1026612, "max": 1026612}, "Number of Batches Since Last Reset": {"sum": 51331.0, "count": 1, "min": 51331, "max": 51331}}}
[04/06/2023 21:56:59 INFO 139636117940032] #throughput_metric: host=algo-1, train throughput=3018.6950032191776 records/second
[2023-04-06 22:02:38.867] [tensorio] [info] epoch_stats={"data_pipeline": "/opt/ml/input/data/train", "epoch": 7, "duration": 338935, "num_examples": 51331, "num_bytes": 147832128}
```

#metrics {"StartTime": 1680818558.8679452, "EndTime": 1680818558.86799, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 2, "model": 0}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30634282467403817, "count": 1, "min": 0.30634282467403817, "max": 0.30634282467403817}}}
#metrics {"StartTime": 1680818558.8680828, "EndTime": 1680818558.8680978, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 2, "model": 1}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.306342971598965, "count": 1, "min": 0.306342971598965, "max": 0.306342971598965}}}
#metrics {"StartTime": 1680818558.8681297, "EndTime": 1680818558.8681383, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 2, "model": 2}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30651108320544806, "count": 1, "min": 0.30651108320544806, "max": 0.30651108320544806}}}
#metrics {"StartTime": 1680818558.868163, "EndTime": 1680818558.8681703, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 2, "model": 3}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3065103093302322, "count": 1, "min": 0.3065103093302322, "max": 0.3065103093302322}}}
#metrics {"StartTime": 1680818558.8682067, "EndTime": 1680818558.8682213, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 2, "model": 4}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30634316362892006, "count": 1, "min": 0.30634316362892006, "max": 0.30634316362892006}}}
#metrics {"StartTime": 1680818558.868293, "EndTime": 1680818558.8683097, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 2, "model": 5}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3063425568869628, "count": 1, "min": 0.3063425568869628, "max": 0.3063425568869628}}}

```
#metrics {"StartTime": 1680818558.8683534, "EndTime": 1680818558.8683684,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 2, "model": 6}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.3065110138902278, "count": 1, "min":
0.3065110138902278, "max": 0.3065110138902278}}}
#metrics {"StartTime": 1680818558.868416, "EndTime": 1680818558.8684323,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 2, "model": 7}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.3065111651112642, "count": 1, "min":
0.3065111651112642, "max": 0.3065111651112642}}}
#metrics {"StartTime": 1680818558.8684866, "EndTime": 1680818558.8685017,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 2, "model": 8}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.3063600672819497, "count": 1, "min":
0.3063600672819497, "max": 0.3063600672819497}}}
#metrics {"StartTime": 1680818558.868542, "EndTime": 1680818558.8685572,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 2, "model": 9}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.3063630591164828, "count": 1, "min":
0.3063630591164828, "max": 0.3063630591164828}}}
[04/06/2023 22:02:38 INFO 139636117940032] #quality_metric: host=algo-1,
epoch=2, train absolute_loss_objective <loss>=0.30634282467403817
[04/06/2023 22:02:38 INFO 139636117940032] #early_stopping_criteria_metric:
host=algo-1, epoch=2, criteria=absolute_loss_objective,
value=0.3063425568869628
[04/06/2023 22:02:38 INFO 139636117940032] Saving model for epoch: 2
[04/06/2023 22:02:38 INFO 139636117940032] Saved checkpoint to
"/tmp/tmpa8t96wh9/mx-mod-0000.params"
[04/06/2023 22:02:38 INFO 139636117940032] #progress_metric: host=algo-1,
completed 60.0 % of epochs
```

```
#metrics {"StartTime": 1680818219.9327796, "EndTime": 1680818558.8749285,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 2, "Meta": "training_data_iter"}, "Metrics": {"Total
Records Seen": {"sum": 3089876.0, "count": 1, "min": 3089876, "max": 3089876},
"Total Batches Seen": {"sum": 154495.0, "count": 1, "min": 154495, "max":
154495}, "Max Records Seen Between Resets": {"sum": 1026612.0, "count": 1,
"min": 1026612, "max": 1026612}, "Max Batches Seen Between Resets": {"sum":
51331.0, "count": 1, "min": 51331, "max": 51331}, "Reset Count": {"sum": 5.0,
"count": 1, "min": 5, "max": 5}, "Number of Records Since Last Reset": {"sum":
1026612.0, "count": 1, "min": 1026612, "max": 1026612}, "Number of Batches Since
Last Reset": {"sum": 51331.0, "count": 1, "min": 51331, "max": 51331}}}
[04/06/2023 22:02:38 INFO 139636117940032] #throughput_metric: host=algo-1,
train throughput=3028.8697577072 records/second
[2023-04-06 22:08:17.597] [tensorio] [info] epoch_stats={"data_pipeline":
"/opt/ml/input/data/train", "epoch": 9, "duration": 338717, "num_examples":
51331, "num_bytes": 147832128}
#metrics {"StartTime": 1680818897.5974853, "EndTime": 1680818897.5975509,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 3, "model": 0}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.30634282800646384, "count": 1,
"min": 0.30634282800646384, "max": 0.30634282800646384}}}
#metrics {"StartTime": 1680818897.597649, "EndTime": 1680818897.5976696,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 3, "model": 1}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.30634271212553577, "count": 1,
"min": 0.30634271212553577, "max": 0.30634271212553577}}}
#metrics {"StartTime": 1680818897.5977187, "EndTime": 1680818897.5977345,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 3, "model": 2}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.30651054514782855, "count": 1,
"min": 0.30651054514782855, "max": 0.30651054514782855}}}
```

#metrics {"StartTime": 1680818897.5977876, "EndTime": 1680818897.5978048, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 3, "model": 3}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30651097490103324, "count": 1, "min": 0.30651097490103324, "max": 0.30651097490103324}}}
#metrics {"StartTime": 1680818897.5978582, "EndTime": 1680818897.5978749, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 3, "model": 4}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30634292670074736, "count": 1, "min": 0.30634292670074736, "max": 0.30634292670074736}}}
#metrics {"StartTime": 1680818897.5979254, "EndTime": 1680818897.5979402, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 3, "model": 5}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30634271637751953, "count": 1, "min": 0.30634271637751953, "max": 0.30634271637751953}}}
#metrics {"StartTime": 1680818897.5979898, "EndTime": 1680818897.5980065, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 3, "model": 6}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30651080431156447, "count": 1, "min": 0.30651080431156447, "max": 0.30651080431156447}}}
#metrics {"StartTime": 1680818897.5980608, "EndTime": 1680818897.5980768, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 3, "model": 7}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3065101406570998, "count": 1, "min": 0.3065101406570998, "max": 0.3065101406570998}}}
#metrics {"StartTime": 1680818897.5981266, "EndTime": 1680818897.5981417, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 3, "model": 8}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3063607875131014, "count": 1, "min": 0.3063607875131014, "max": 0.3063607875131014}}}

```
#metrics {"StartTime": 1680818897.5981922, "EndTime": 1680818897.5982091,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 3, "model": 9}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.3063626020224778, "count": 1, "min":
0.3063626020224778, "max": 0.3063626020224778}}}
[04/06/2023 22:08:17 INFO 139636117940032] #quality_metric: host=algo-1,
epoch=3, train absolute_loss_objective <loss>=0.30634282800646384
[04/06/2023 22:08:17 INFO 139636117940032] #early_stopping_criteria_metric:
host=algo-1, epoch=3, criteria=absolute_loss_objective,
value=0.30634271212553577
[04/06/2023 22:08:17 INFO 139636117940032] Saving model for epoch: 3
[04/06/2023 22:08:17 INFO 139636117940032] Saved checkpoint to
"/tmp/tmp9bjdn5nx/mx-mod-0000.params"
[04/06/2023 22:08:17 INFO 139636117940032] #progress_metric: host=algo-1,
completed 80.0 % of epochs
#metrics {"StartTime": 1680818558.8800168, "EndTime": 1680818897.6050825,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 3, "Meta": "training_data_iter"}, "Metrics": {"Total
Records Seen": {"sum": 4116488.0, "count": 1, "min": 4116488, "max": 4116488},
"Total Batches Seen": {"sum": 205826.0, "count": 1, "min": 205826, "max":
205826}, "Max Records Seen Between Resets": {"sum": 1026612.0, "count": 1,
"min": 1026612, "max": 1026612}, "Max Batches Seen Between Resets": {"sum":
51331.0, "count": 1, "min": 51331, "max": 51331}, "Reset Count": {"sum": 6.0,
"count": 1, "min": 6, "max": 6}, "Number of Records Since Last Reset": {"sum":
1026612.0, "count": 1, "min": 1026612, "max": 1026612}, "Number of Batches Since
Last Reset": {"sum": 51331.0, "count": 1, "min": 51331, "max": 51331}}}
[04/06/2023 22:08:17 INFO 139636117940032] #throughput_metric: host=algo-1,
train throughput=3030.810793645813 records/second
[2023-04-06 22:13:56.338] [tensorio] [info] epoch_stats={"data_pipeline":
"/opt/ml/input/data/train", "epoch": 11, "duration": 338731, "num_examples":
51331, "num_bytes": 147832128}
```

#metrics {"StartTime": 1680819236.3384461, "EndTime": 1680819236.3385057, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 4, "model": 0}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30634272489820846, "count": 1, "min": 0.30634272489820846, "max": 0.30634272489820846}}}
#metrics {"StartTime": 1680819236.338594, "EndTime": 1680819236.3386135, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 4, "model": 1}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3063427838327219, "count": 1, "min": 0.3063427838327219, "max": 0.3063427838327219}}}
#metrics {"StartTime": 1680819236.3386617, "EndTime": 1680819236.338676, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 4, "model": 2}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30651035140950894, "count": 1, "min": 0.30651035140950894, "max": 0.30651035140950894}}}
#metrics {"StartTime": 1680819236.338726, "EndTime": 1680819236.3387418, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 4, "model": 3}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.30651057177518465, "count": 1, "min": 0.30651057177518465, "max": 0.30651057177518465}}}
#metrics {"StartTime": 1680819236.3387895, "EndTime": 1680819236.3388047, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 4, "model": 4}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3063428473668839, "count": 1, "min": 0.3063428473668839, "max": 0.3063428473668839}}}
#metrics {"StartTime": 1680819236.3388567, "EndTime": 1680819236.338873, "Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation": "training", "epoch": 4, "model": 5}, "Metrics": {"train_absolute_loss_objective": {"sum": 0.3063427172287988, "count": 1, "min": 0.3063427172287988, "max": 0.3063427172287988}}}

```
#metrics {"StartTime": 1680819236.3389242, "EndTime": 1680819236.3389404,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 4, "model": 6}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.30651045544521865, "count": 1,
"min": 0.30651045544521865, "max": 0.30651045544521865}}}
#metrics {"StartTime": 1680819236.3389888, "EndTime": 1680819236.3390043,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 4, "model": 7}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.3065108229316003, "count": 1, "min":
0.3065108229316003, "max": 0.3065108229316003}}}
#metrics {"StartTime": 1680819236.3390574, "EndTime": 1680819236.3390727,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 4, "model": 8}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.30636111502349667, "count": 1,
"min": 0.30636111502349667, "max": 0.30636111502349667}}}
#metrics {"StartTime": 1680819236.3391252, "EndTime": 1680819236.3391407,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 4, "model": 9}, "Metrics":
{"train_absolute_loss_objective": {"sum": 0.30636256194395695, "count": 1,
"min": 0.30636256194395695, "max": 0.30636256194395695}}}
[04/06/2023 22:13:56 INFO 139636117940032] #quality_metric: host=algo-1,
epoch=4, train absolute_loss_objective <loss>=0.30634272489820846
[04/06/2023 22:13:56 INFO 139636117940032] #early_stopping_criteria_metric:
host=algo-1, epoch=4, criteria=absolute_loss_objective,
value=0.3063427172287988
[04/06/2023 22:13:56 INFO 139636117940032] Saving model for epoch: 4
[04/06/2023 22:13:56 INFO 139636117940032] Saved checkpoint to
"/tmp/tmprqo8bemm/mx-mod-0000.params"
[04/06/2023 22:13:56 INFO 139636117940032] #progress_metric: host=algo-1,
completed 100.0 % of epochs
```

```
#metrics {"StartTime": 1680818897.6063855, "EndTime": 1680819236.346007,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training", "epoch": 4, "Meta": "training_data_iter"}, "Metrics": {"Total
Records Seen": {"sum": 5143100.0, "count": 1, "min": 5143100, "max": 5143100},
"Total Batches Seen": {"sum": 257157.0, "count": 1, "min": 257157, "max":
257157}, "Max Records Seen Between Resets": {"sum": 1026612.0, "count": 1,
"min": 1026612, "max": 1026612}, "Max Batches Seen Between Resets": {"sum":
51331.0, "count": 1, "min": 51331, "max": 51331}, "Reset Count": {"sum": 7.0,
"count": 1, "min": 7, "max": 7}, "Number of Records Since Last Reset": {"sum":
1026612.0, "count": 1, "min": 1026612, "max": 1026612}, "Number of Batches Since
Last Reset": {"sum": 51331.0, "count": 1, "min": 51331, "max": 51331}}}
[04/06/2023 22:13:56 INFO 139636117940032] #throughput_metric: host=algo-1,
train throughput=3030.6806190282455 records/second
[04/06/2023 22:13:56 WARNING 139636117940032] wait_for_all_workers will not
sync workers since the kv store is not running distributed
[04/06/2023 22:13:56 WARNING 139636117940032] wait_for_all_workers will not
sync workers since the kv store is not running distributed
[2023-04-06 22:13:56.348] [tensorio] [info] epoch_stats={"data_pipeline":
"/opt/ml/input/data/train", "epoch": 13, "duration": 0, "num_examples": 1,
"num_bytes": 2880}
[2023-04-06 22:18:50.757] [tensorio] [info] epoch_stats={"data_pipeline":
"/opt/ml/input/data/train", "epoch": 15, "duration": 294406, "num_examples":
51331, "num_bytes": 147832128}
[04/06/2023 22:18:50 INFO 139636117940032] #train_score (algo-1) :
('absolute_loss_objective', 7399.774961673081)
[04/06/2023 22:18:50 INFO 139636117940032] #train_score (algo-1) : ('mse',
135367573.97617796)
[04/06/2023 22:18:50 INFO 139636117940032] #train_score (algo-1) :
('absolute_loss', 7399.774961673081)
[04/06/2023 22:18:50 INFO 139636117940032] #train_score (algo-1) : ('rmse',
11634.75715157725)
[04/06/2023 22:18:50 INFO 139636117940032] #train_score (algo-1) : ('r2',
0.7670055688981828)
[04/06/2023 22:18:50 INFO 139636117940032] #train_score (algo-1) : ('mae',
7399.774962383461)
```

```
[04/06/2023 22:18:50 INFO 139636117940032] #quality_metric: host=algo-1,
train absolute_loss_objective <loss>=7399.774961673081
[04/06/2023 22:18:50 INFO 139636117940032] #quality_metric: host=algo-1,
train mse <loss>=135367573.97617796
[04/06/2023 22:18:50 INFO 139636117940032] #quality_metric: host=algo-1,
train absolute_loss <loss>=7399.774961673081
[04/06/2023 22:18:50 INFO 139636117940032] #quality_metric: host=algo-1,
train rmse <loss>=11634.75715157725
[04/06/2023 22:18:50 INFO 139636117940032] #quality_metric: host=algo-1,
train r2 <loss>=0.7670055688981828
[04/06/2023 22:18:50 INFO 139636117940032] #quality_metric: host=algo-1,
train mae <loss>=7399.774962383461
[04/06/2023 22:18:50 INFO 139636117940032] Best model found for
hyperparameters: {"optimizer": "adam", "learning_rate": 0.005, "l1": 0.0, "wd":
0.0001, "lr_scheduler_step": 10, "lr_scheduler_factor": 0.99,
"lr_scheduler_minimum_lr": 1e-05}
[04/06/2023 22:18:50 INFO 139636117940032] Saved checkpoint to
"/tmp/tmps8gf05dt/mx-mod-0000.params"
[04/06/2023 22:18:50 INFO 139636117940032] Test data is not provided.
#metrics {"StartTime": 1680817528.6603682, "EndTime": 1680819530.78087,
"Dimensions": {"Algorithm": "Linear Learner", "Host": "algo-1", "Operation":
"training"}, "Metrics": {"initialize.time": {"sum": 3379.92787361145, "count":
1, "min": 3379.92787361145, "max": 3379.92787361145}, "epochs": {"sum": 5.0,
"count": 1, "min": 5, "max": 5}, "check_early_stopping.time": {"sum":
2.8939247131347656, "count": 5, "min": 0.1914501190185547, "max":
1.300811767578125}, "update.time": {"sum": 1704269.9975967407, "count": 5,
"min": 338722.2228050232, "max": 347789.7119522095}, "finalize.time": {"sum":
294430.43327331543, "count": 1, "min": 294430.43327331543, "max":
294430.43327331543}, "setuptime": {"sum": 2.469778060913086, "count": 1, "min":
2.469778060913086, "max": 2.469778060913086}, "totaltime": {"sum":
2002230.4542064667, "count": 1, "min": 2002230.4542064667, "max":
2002230.4542064667}}}

2023-04-06 22:19:08 Uploading - Uploading generated training model
2023-04-06 22:19:08 Completed - Training job completed
Training seconds: 2141
```

```
Billable seconds: 2141
```

Endpoint creation & Model evaluation

[114]:
```python
linear_regressor = linear.deploy(initial_instance_count = 1, instance_type =
 ↪'ml.m4.xlarge')
```

```
INFO:sagemaker:Creating model with name: linear-learner-2023-04-06-22-28-34-861
INFO:sagemaker:Creating endpoint-config with name linear-
learner-2023-04-06-22-28-34-861
INFO:sagemaker:Creating endpoint with name linear-
learner-2023-04-06-22-28-34-861

--------!
```

[111]:
```python
from sagemaker.predictor import csv_serializer, json_deserializer
```

[118]:
```python
linear_regressor.serializer = csv_serializer
linear_regressor.deserializer = json_deserializer
```

I get (413) errors when the test dataset is too large, so I broke them down into chunks of 20,000 each

[137]:
```python
result0 = linear_regressor.predict(X_test_sc[0:20000])
result1 = linear_regressor.predict(X_test_sc[20000:40000])
result2 = linear_regressor.predict(X_test_sc[40000:60000])
result3 = linear_regressor.predict(X_test_sc[60000:80000])
result4 = linear_regressor.predict(X_test_sc[80000:100000])

result5 = linear_regressor.predict(X_test_sc[100000:120000])
result6 = linear_regressor.predict(X_test_sc[120000:140000])
result7 = linear_regressor.predict(X_test_sc[140000:160000])
result8 = linear_regressor.predict(X_test_sc[160000:180000])
result9 = linear_regressor.predict(X_test_sc[180000:200000])

result10 = linear_regressor.predict(X_test_sc[200000:220000])
result11 = linear_regressor.predict(X_test_sc[220000:240000])
result12 = linear_regressor.predict(X_test_sc[240000:260000])
result13 = linear_regressor.predict(X_test_sc[260000:280000])
result14 = linear_regressor.predict(X_test_sc[280000:300000])

result15 = linear_regressor.predict(X_test_sc[300000:320000])
result16 = linear_regressor.predict(X_test_sc[320000:340000])
result17 = linear_regressor.predict(X_test_sc[340000:360000])
result18 = linear_regressor.predict(X_test_sc[360000:380000])
result19 = linear_regressor.predict(X_test_sc[380000:400000])

result20 = linear_regressor.predict(X_test_sc[400000:420000])
result21 = linear_regressor.predict(X_test_sc[420000:439977])
```

```
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

```
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

```
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The csv_serializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
WARNING:sagemaker.deprecations:The json_deserializer has been renamed in
sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

[136]:
```
len(X_test_sc)
```

[136]: 439977

[143]:
```
predictions0 = np.array([res['score'] for res in result0['predictions']])
predictions1 = np.array([res['score'] for res in result1['predictions']])
predictions2 = np.array([res['score'] for res in result2['predictions']])
predictions3 = np.array([res['score'] for res in result3['predictions']])
predictions4 = np.array([res['score'] for res in result4['predictions']])
predictions5 = np.array([res['score'] for res in result5['predictions']])
```

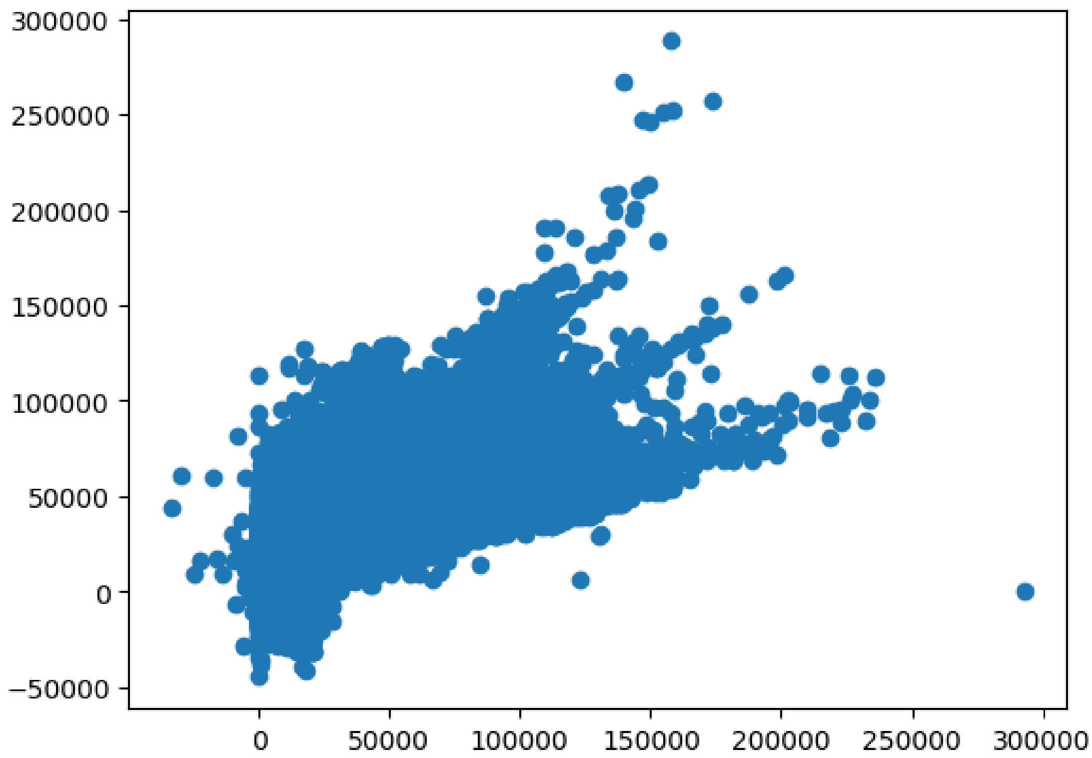
```python
predictions6 = np.array([res['score'] for res in result6['predictions']])
predictions7 = np.array([res['score'] for res in result7['predictions']])
predictions8 = np.array([res['score'] for res in result8['predictions']])
predictions9 = np.array([res['score'] for res in result9['predictions']])
predictions10 = np.array([res['score'] for res in result10['predictions']])
predictions11 = np.array([res['score'] for res in result11['predictions']])
predictions12 = np.array([res['score'] for res in result12['predictions']])
predictions13 = np.array([res['score'] for res in result13['predictions']])
predictions14 = np.array([res['score'] for res in result14['predictions']])
predictions15 = np.array([res['score'] for res in result15['predictions']])
predictions16 = np.array([res['score'] for res in result16['predictions']])
predictions17 = np.array([res['score'] for res in result17['predictions']])
predictions18 = np.array([res['score'] for res in result18['predictions']])
predictions19 = np.array([res['score'] for res in result19['predictions']])
predictions20 = np.array([res['score'] for res in result20['predictions']])
predictions21 = np.array([res['score'] for res in result21['predictions']])
```

[144]:
```python
#predictions = np.array([res['score'] for res in result['predictions']])
```

[149]:
```python
all_preds = np.concatenate((predictions0,
               predictions1,
               predictions2,
               predictions3,
               predictions4,
               predictions5,
               predictions6,
               predictions7,
               predictions8,
               predictions9,
               predictions10,
               predictions11,
               predictions12,
               predictions13,
               predictions14,
               predictions15,
               predictions16,
               predictions17,
               predictions18,
               predictions19,
               predictions20,
               predictions21))
```

[150]:
```python
plt.scatter(y_test, all_preds)
```

[150]:
```
<matplotlib.collections.PathCollection at 0x7f327f2dbb10>
```



[132]:
```
from sklearn import metrics
```

[151]:
```
np.sqrt(metrics.mean_squared_error(y_test, all_preds))
```

[151]: 11592.010286298273

[155]:
```
%store all_preds
```

Stored 'all_preds' (ndarray)

[156]:
```
%store y_test
```

Stored 'y_test' (Series)

[157]:
```
%store X_test_sc
```

Stored 'X_test_sc' (ndarray)

[158]:
```
%store y_train
```

Stored 'y_train' (Series)

[159]:
```
%store X_train_sc
```

```
Stored 'X_train_sc' (ndarray)
```

[ ]:

```

```