THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

VISUALIZATION OF SAN FRANCISCO PAYROLL

By: Osemekhian Ehilen

Final Project

12/17/2022

TABLE OF CONTENT

Contents

ABSTRACT

This project looked at analyzing and visualizing San Francisco Payroll between the year 2011 through 2019. Python programming language was leveraged alongside packages like matplotlib, seaborn, numpy, pandas and dash amongst other to analyze and visualize various charts to tell the story of San Francisco Payroll. A major observation was that full-time workers are in high demand in San Francisco with typical base pay between $100k-$150k.

INTRODUCTION

In this project, the pay of public servants in San Francisco, United States, with respect to different Job titles and status are going to be analyzed with visual representation.

Python programming language will be the main aide in achieving the above. Nevertheless, a package- Dash will be used to create a website-like dashboard to visualize interactive plots. Visualizations for this report will be generated by the seaborn package in python.

This report will see various sections, such as:

- Dataset Description
- Pre-processing
- Outlier Detection & Removal
- Principal Component Analysis (PCA)
- Normality Test
- Data Transformation
- Heatmap & Person Correlation Coefficient Matrix
- Statistics
- Data Visualization

Description of Dataset

This dataset contains the California public employee salaries from years 2011 to 2019 from kaggle with 357,407 samples.

This dataset is selected to know the various pay across various job types in a tech city- San Francisco. This analysis on pay of diverse job types and status can help one decide which field to opt in due to different reasons that can be incited from this project.

The dataset contain categorical and numerical features as explained from the table below:

Features	Count
Categorical	3
Continuous	7

The feature names are given below with respect to their dependence:

Dependent	Independent	Others
Total Pay & Benefits	Base Pay	Employee Name
	Overtime Pay	Job Title
	Other Pay	Year
	Benefits	Total Pay
	Status	

Pre-Processing

The null values for Status column was replaced with the modal class of not-null items in status column.

Zero value was assigned to continuous columns with “Not Available” entries aside base pay.

Also, I Encoded Status column with 0 for part-time and 1 for full-time.

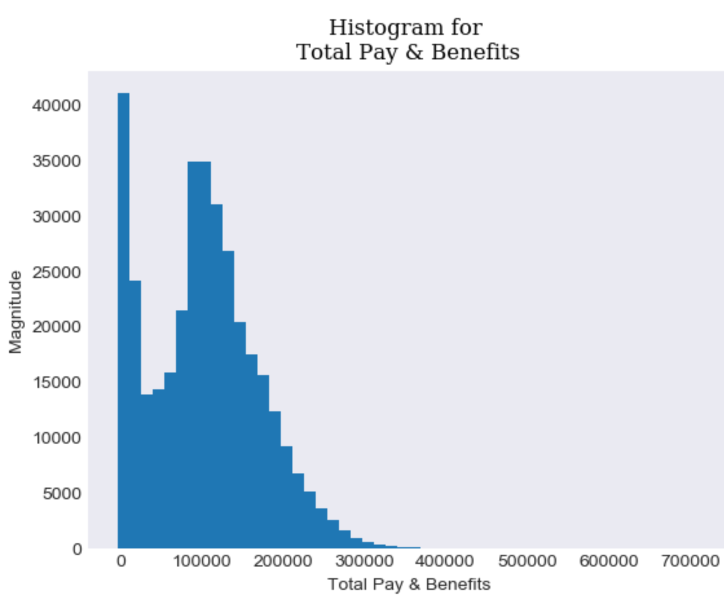
The appropriate data types were assigned to miss-matched columns using pandas dataframe astype method.

A flip was performed to make the “Year” column in ascending sequence.

Outlier Detection & Removal

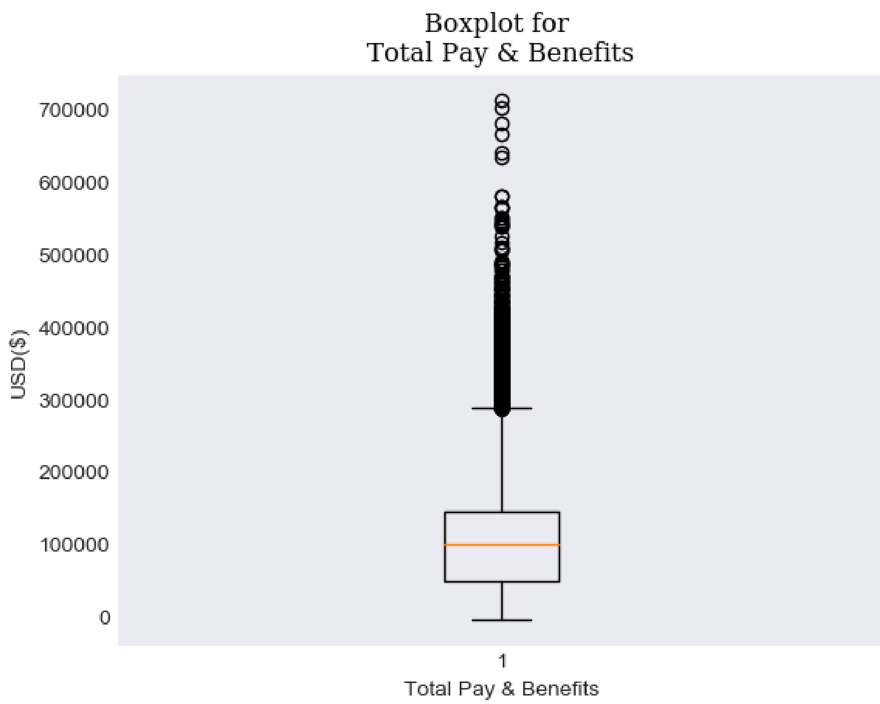
Amongst the various ways of detecting outliers, histogram and boxplot will be leveraged here.

First, we plot the histogram plot for the cleaned dataset (i.e. pre-processed dataset).



The histogram above is skewed to the right and the bell shape of a Normal data is not evident. This is shows that outliers exist in this dataset.

Secondly, the boxplot generated shows data points above the top whisker, this also indicates positive or right skewness and not Normal distribution of data points.

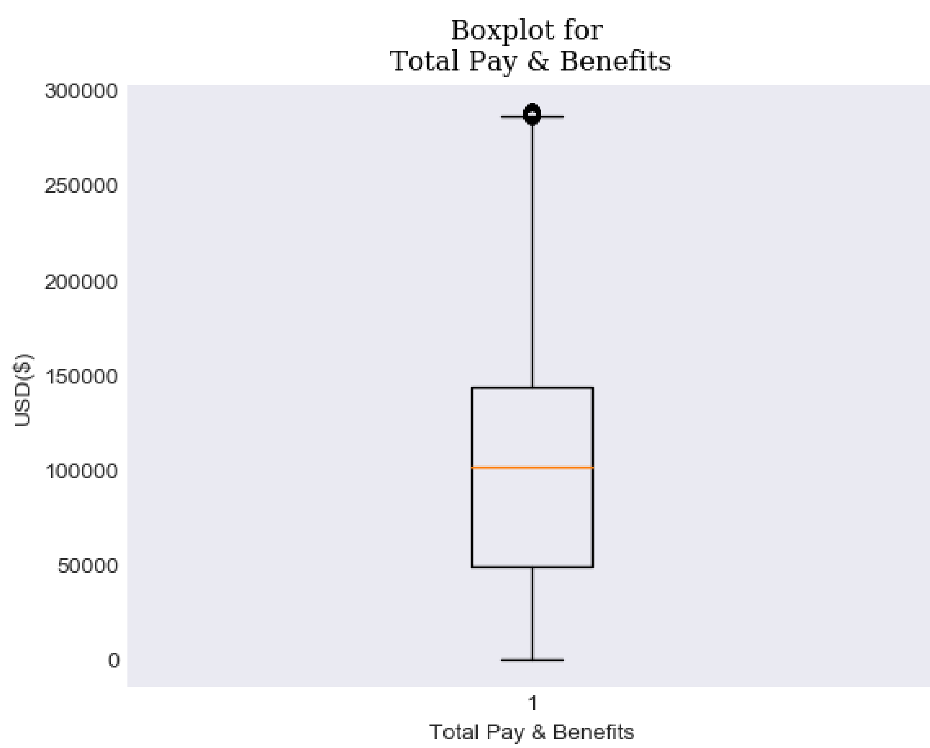


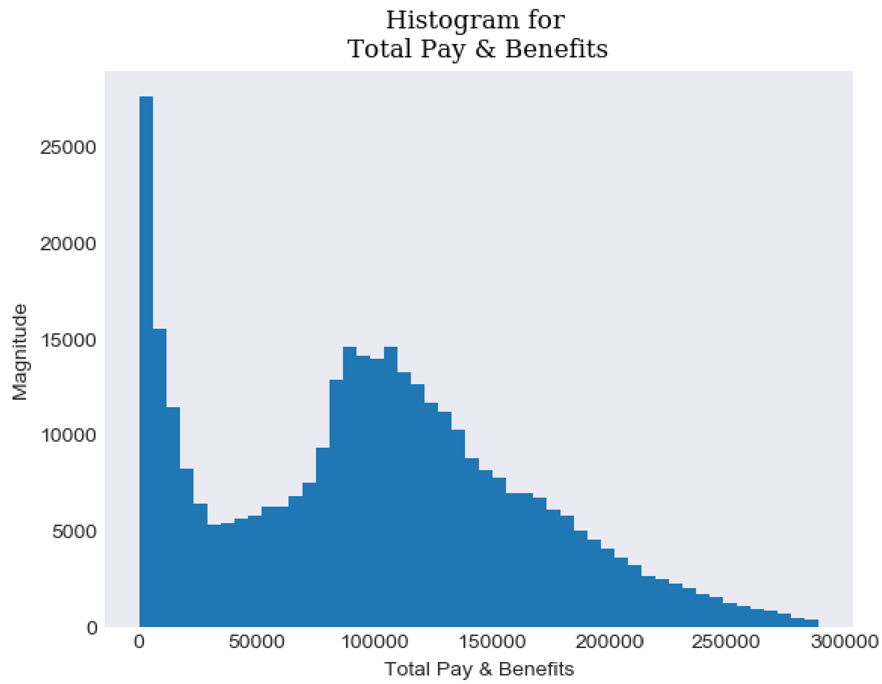
The Interquartile Range (IQR) will be used to remove these outliers. The IQR is the difference between the 75th and 25th percentiles (IQR= $Q_3 - Q_1$) which when applied to filter the dataset, it trims out data point beyond 1.5*IQR lower and upper bound range (i.e. whiskers).

The code below shows the implementation of the above algorithm:

```python
def outlier(data):
  global Q1,Q3
  sorted(data)
  Q1,Q3 = np.percentile(data , [25,75])
  IQR = Q3-Q1
  lower_range = Q1 - (1.5 * IQR)
  upper_range = Q3 + (1.5 * IQR)
  return lower_range,upper_range
```

After performing the IQR to remove outliers, we can see the box plot looking a lot better than its initial.





Also, the histogram plot spreads somewhat evenly on both sides.

Principal Component Analysis (PCA)

The PCA is popular in its ability to reduce dimensions and still explain significant portion of a dataset.

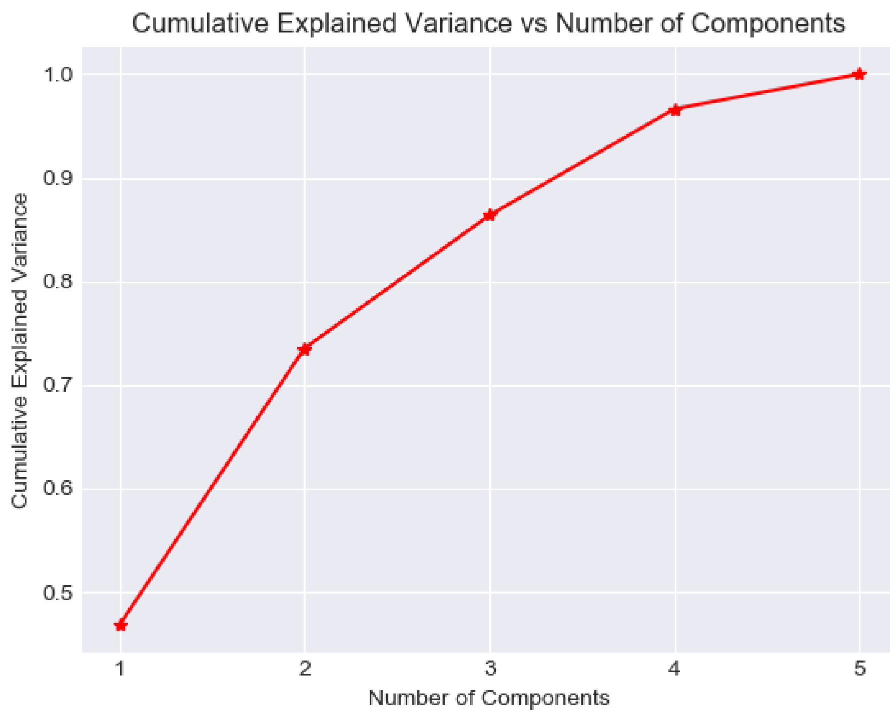
The singular values are generated below. The closer a feature's value the more likely there exist collinearity in amongst features. In this case Status feature shows evidence of collinearity.

	Singular Values
Base Pay	2.7085e+15
Overtime Pay	5.421e+13
Other Pay	3.25042e+13
Other Pay	2.45144e+13
Benefits	67285.9
Status	0.00142788

The condition number also helps us track collinearity. As we see below, the condition number is very large and far beyond 1000. This indicates strong degree of collinearity present amongst features of dataset.

	Condition Number
0	7.38273e+15

Upon performing a PCA, an explained variance is generated below:



The explained variance tells that 5 features from the given 6 features can still tell 100% of the dataset variance in other words 5 features instead of 6 can give 100% of information of what 6 features will give.

I removed Status feature since it had the least singular value. After that I generated singular values and condition number with 5 features.

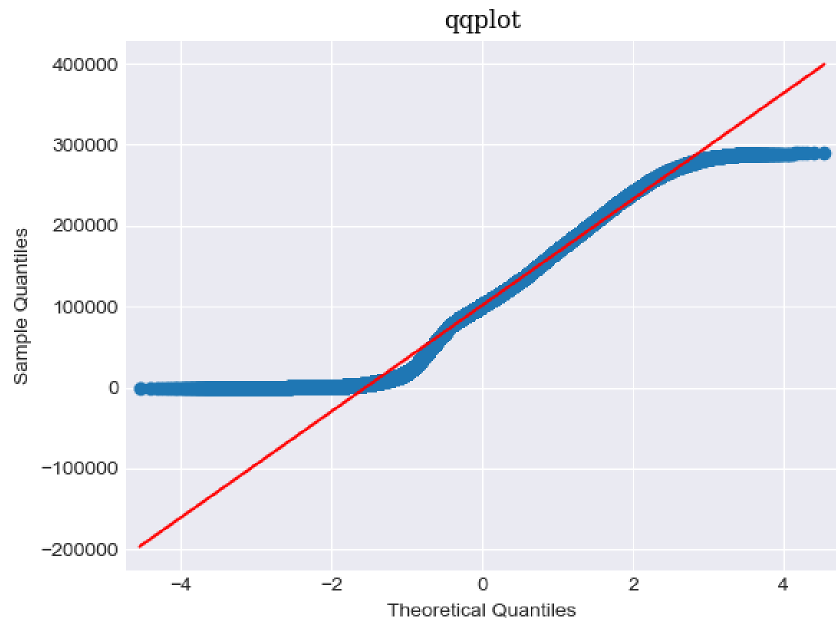
	Singular Values
0	994622
1	568540
2	273223
3	218504
4	71504.2

	Condition Number
0	3.72961

The new singular values are very far from zero which indicates no degree of collinearity and condition number less than 10 which indicates very weak degree of collinearity.

Normality Test

The QQ-Plot and some statistical test will be leveraged in testing for Normality.



The QQ-plot above shows that with the quantiles not aligned on the angle 45 line the dataset is not Normal.

Shapiro, Kolmogorov-Smirnov and D’Agonisto statistical test will be performed to test for Normality. The test results are given below:

```
Shapiro test : Total Pay & Benefits dataset : statistics = 0.97 p-vlaue of =0.00
Shapiro test: Total Pay & Benefits dataset is NOT Normal
==================================================
None
==================================================
K-S test: Total Pay & Benefits dataset: statistics= 0.06 p-value = 0.00
K-S test : Total Pay & Benefits dataset is Not Normal
==================================================
None
==================================================
da_k_squared test: Total Pay & Benefits dataset: statistics= 12126.20 p-value = 0.00
da_k_squared test : Total Pay & Benefits dataset is Not Normal
==================================================
None
```

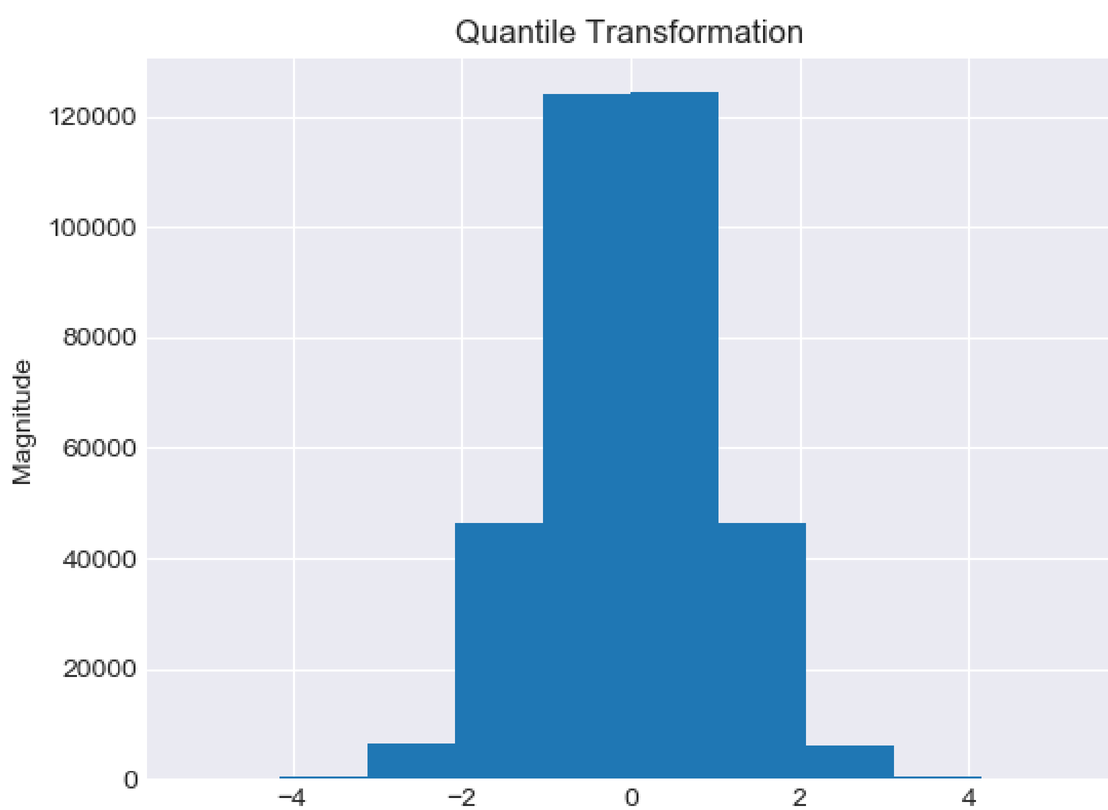
The graphical and statistical tests agree to not Normal behavior of the dependent variable.

Data Transformation

Data Transformation is done to make the distribution of data to be Normal because most algorithms require data from a normal distribution.

For this project I chose the Quantile Transformer from the sci-kit learn package.

After performing a Quantile Transformation, we see the histogram plot below:

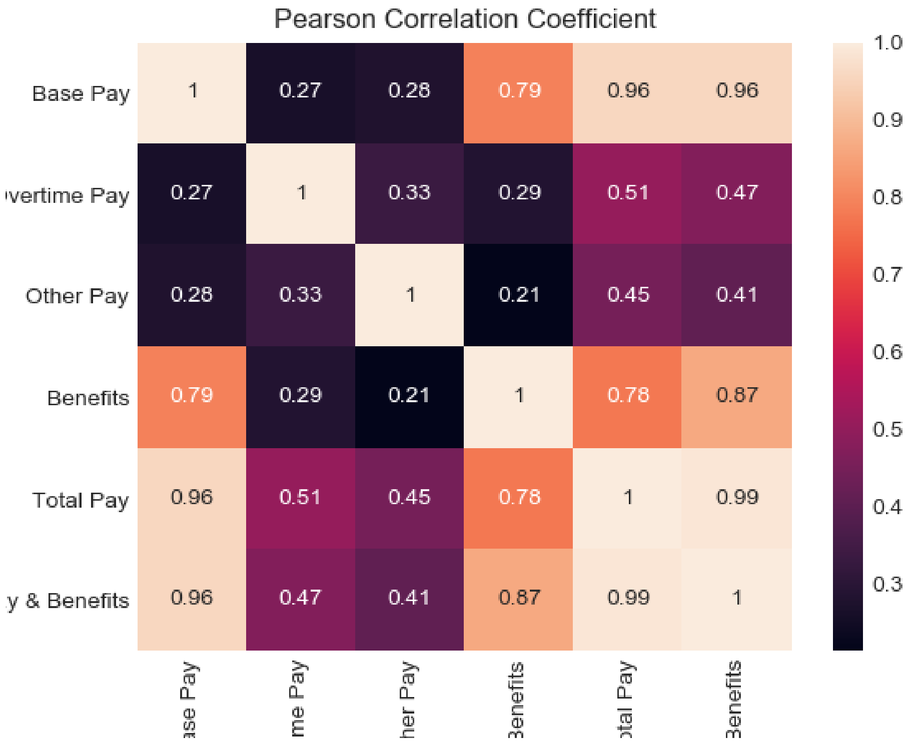


The three statistical test shows the result with only Kolmogorov-Smirnov indicating normality with p-value greater than 0.01.

```
==================================================
Shapiro test : Total Pay & Benefits dataset : statistics = 1.00 p-vlaue of =0.00
Shapiro test: Total Pay & Benefits dataset is NOT Normal
==================================================
None
==================================================
K-S test: Total Pay & Benefits dataset: statistics= 0.00 p-value = 0.16
K-S test:  Total Pay & Benefits dataset is Normal
==================================================
None
==================================================
da_k_squared test: Total Pay & Benefits dataset: statistics= 29.44 p-value = 0.00
da_k_squared test : Total Pay & Benefits dataset is Not Normal
==================================================
```

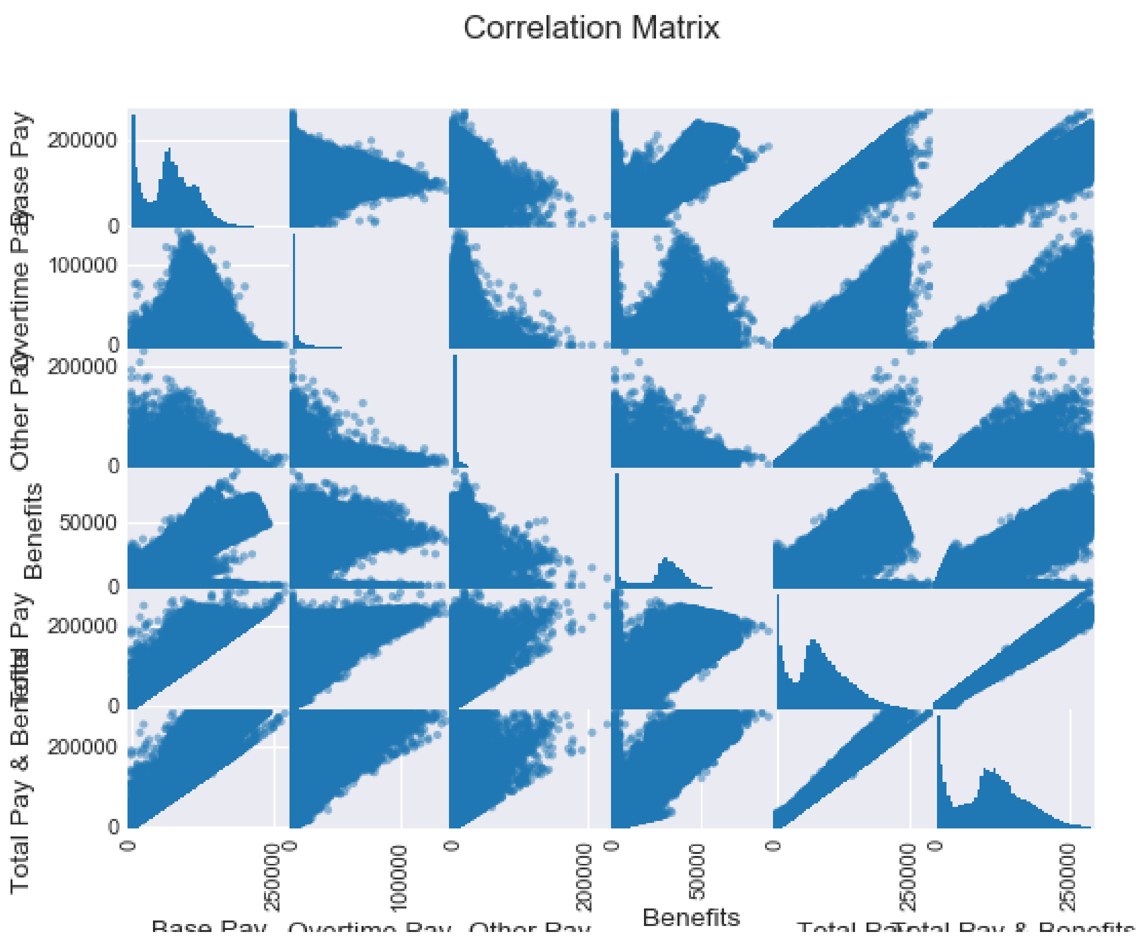
Heatmap & Pearson Correlation Coefficient Matrix

The heatmap displayed below visualizes the correlation between numerical features present in this dataset:



The dependent variable- Total Pay & Benefits has the strongest positive correlation with Total Pay (99%), while the least positive correlation with Other Pay (41%).

The correlation matrix using scatter matrix plot below also indicate similar behavior as the heatmap above.



Statistics

Basic measure of central tendency such as mean and median were generated for the independent variables.

Statistics	mean	median
Base Pay	69818.2	68326.5
Overtime Pay	5657.26	0
Other Pay	3355.51	704.7
Benefits	22534.5	27190.7
Total Pay	78830.9	75738.3
Total Pay & Benefits	101365	101514

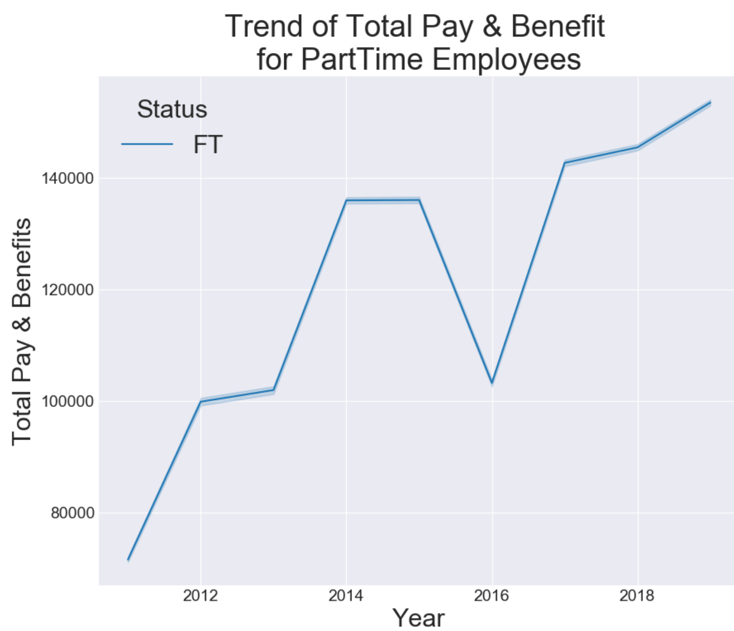
The variation in mean amongst features can indicate that the features are from different scales. Since the median provides a better representation of a value and having nothing to do with skewness; we can for example note that overtime pay was not really gotten most workers.

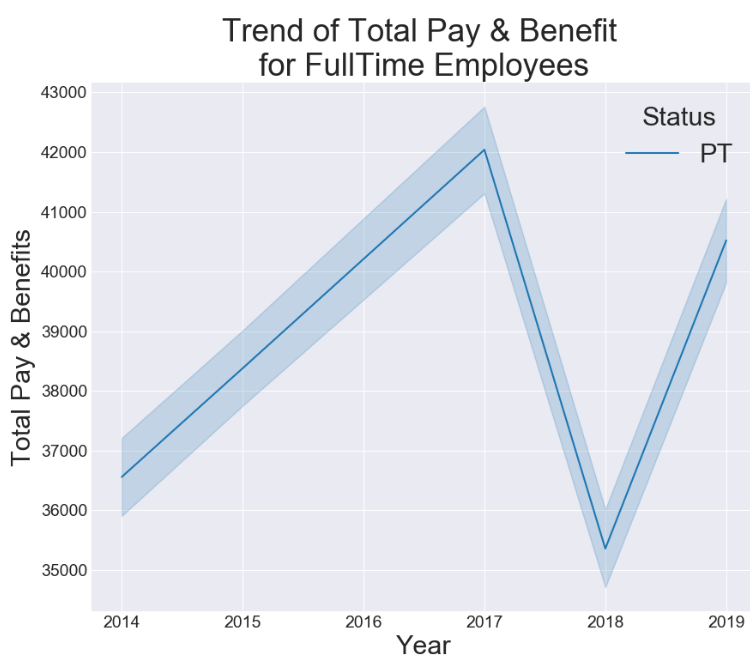
Also, the typical Base Pay for workers is $68,326.50.

Data Visualization

Line Plot

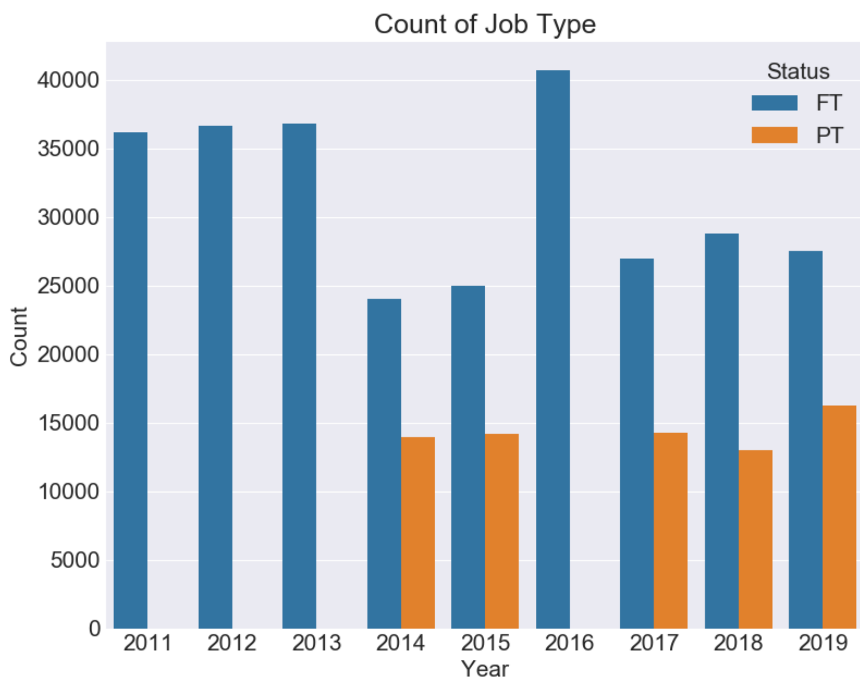
The line plot below shows an upward trend for full-time workers Total Pay & Benefits over the year's while an irregular trend (zig-zag) for part-time workers.





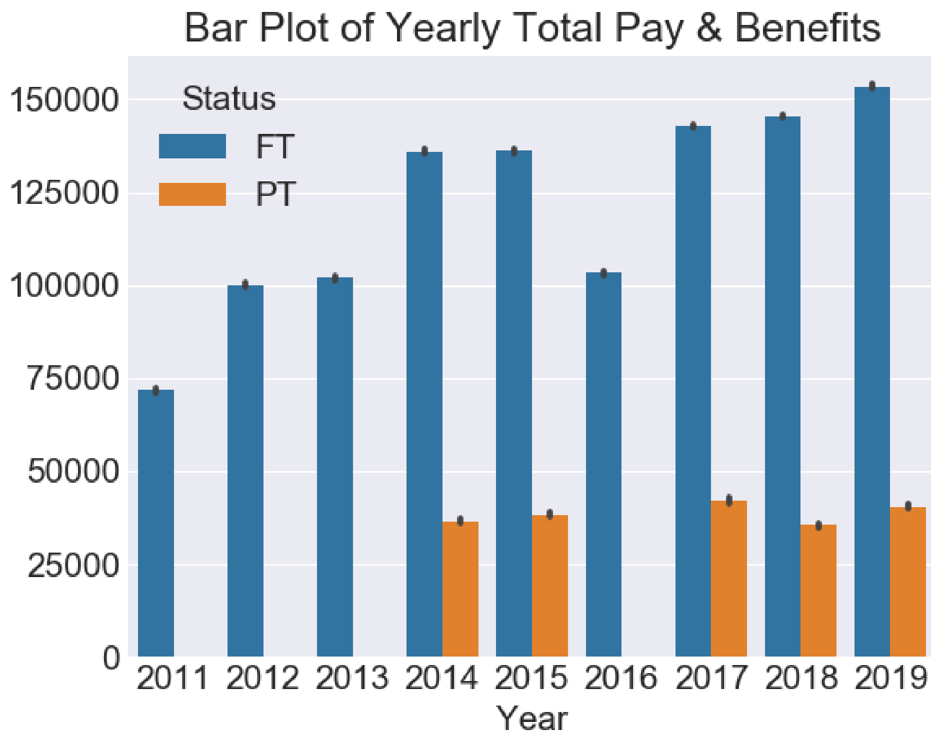
Count Plot

The count plot below indicates the number of workers (full/part time) over the years. Part-time workers can be seen to exist from 2014-2015, 2017-2019. Employers may be highly interested in full-time workers.



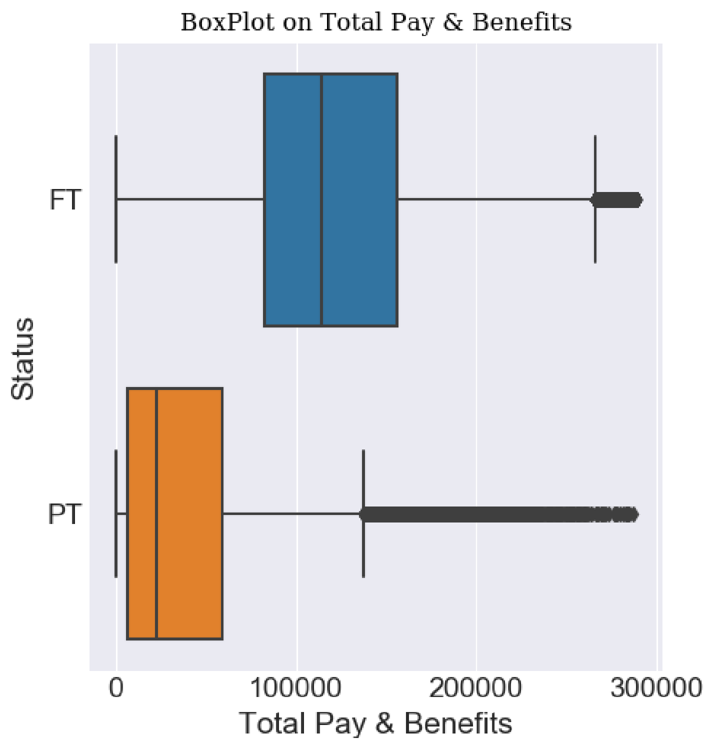
Bar Plot

The bar plot below gives the Total Pay & Benefits of full/part time workers over the years. Part-time workers earned less than $50,000 over the years (2011-2019).



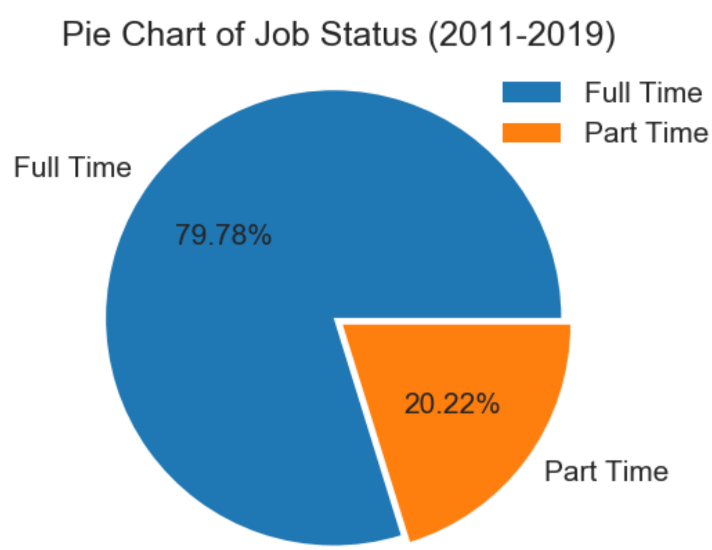
Category Plot

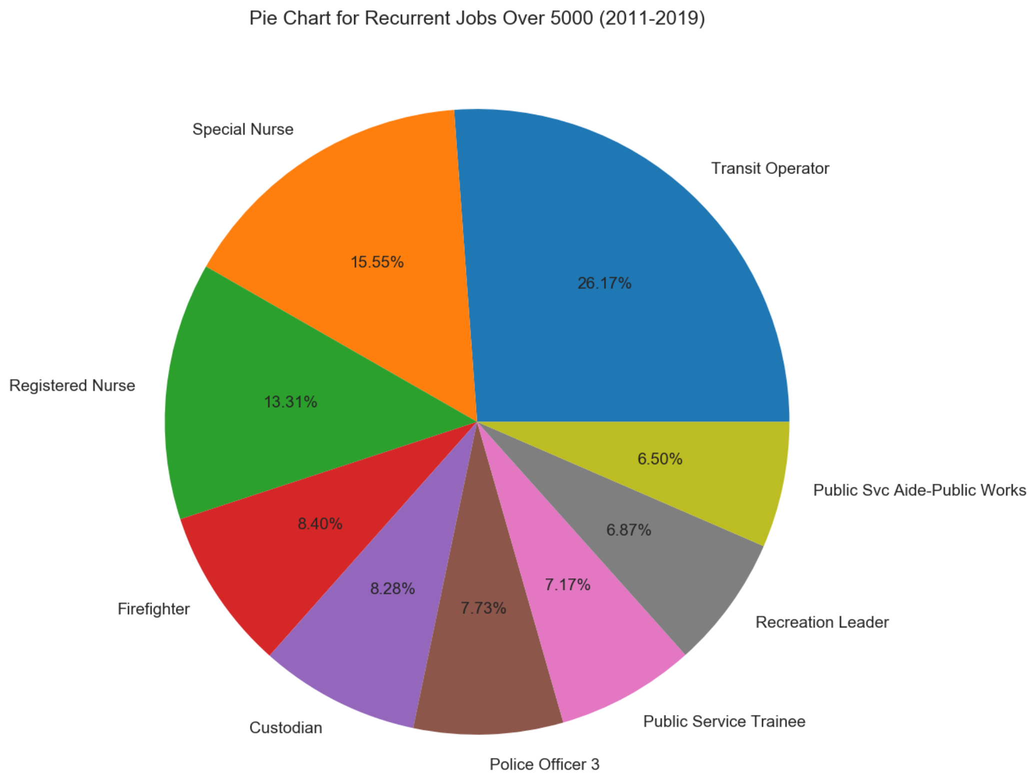
The category plot below shows us that 75% of part-time Total Pay & Benefits are less than 25% of full-time's value. The geographic area could be focusing mainly on full-time workers.



Pie Chart

Over the years 79.78% of Total Pay & Benefits are attributed to full-time staffs while 20.225 goes to part-time staffs.

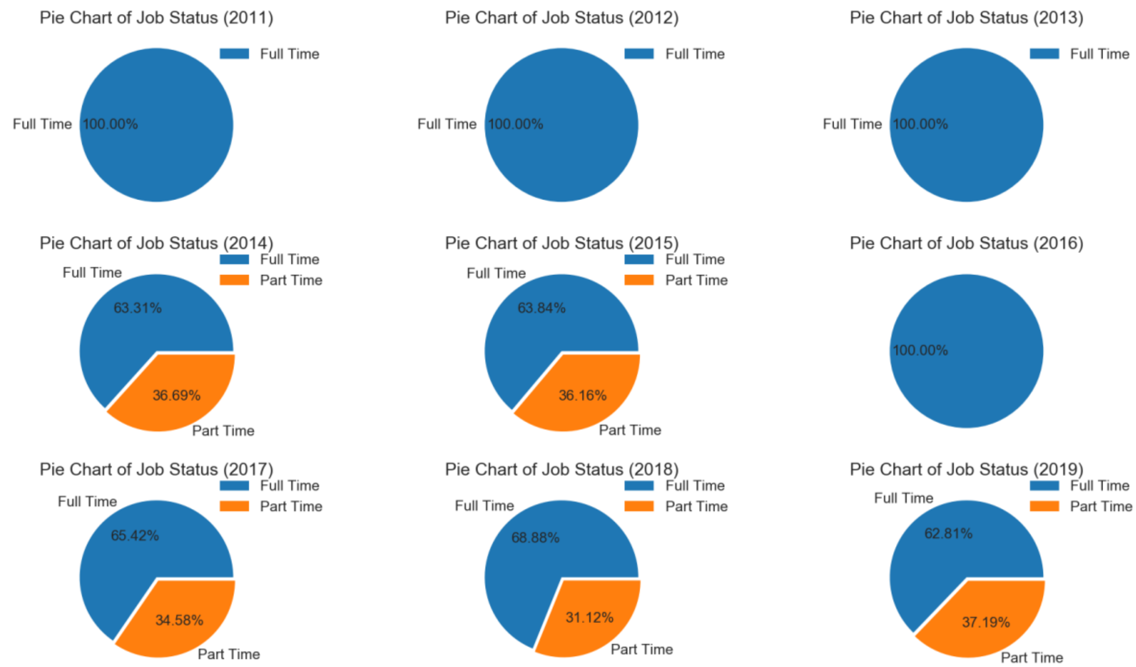




The pie chart above shows that transit Operators have 26.17% of total jobs greater than 5000 followed by Special Nurse, Registered Nurse Fire Firefighters and so on. This location may be good to practice Nursing and get jobs in the transportation sector amongst other observations.

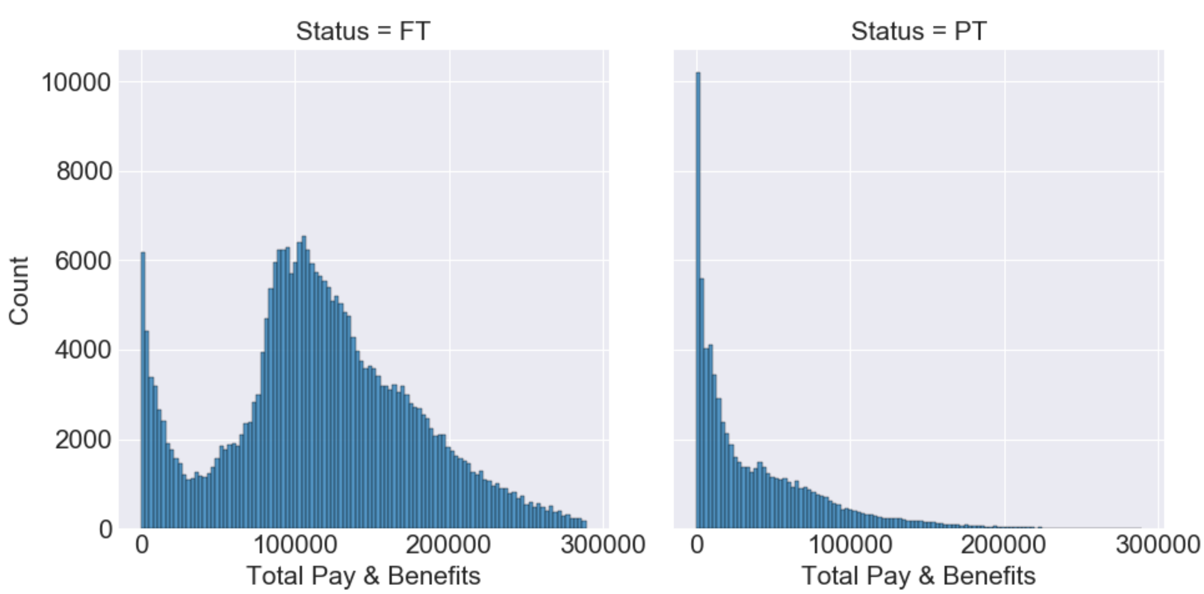
Subplots (Pie Chart)

The subplots below show the percentage of number of FT & PT workers over the years. You can also observe how the part-time workers are relegated in some years and overall have 37.10% of jobs (in 2019).



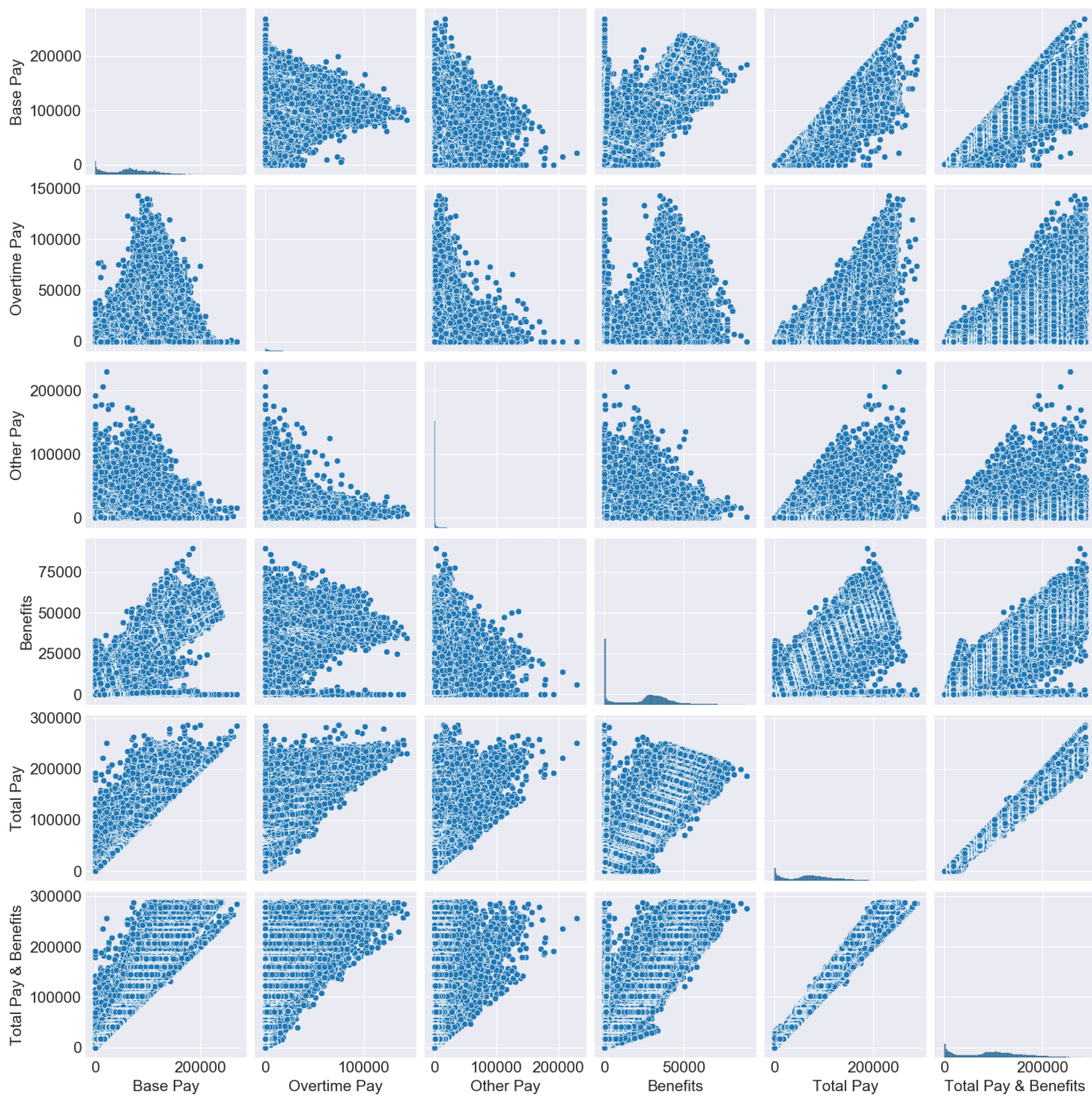
Distribution Plot

The plot below shows the distribution of Total Pay & Benefits between FT and PT workers. PT workers are very skewed to the right compared to FT workers. The typical average pay with benefits for FT is greater than $100,000 compared to PT with less than $50,000.



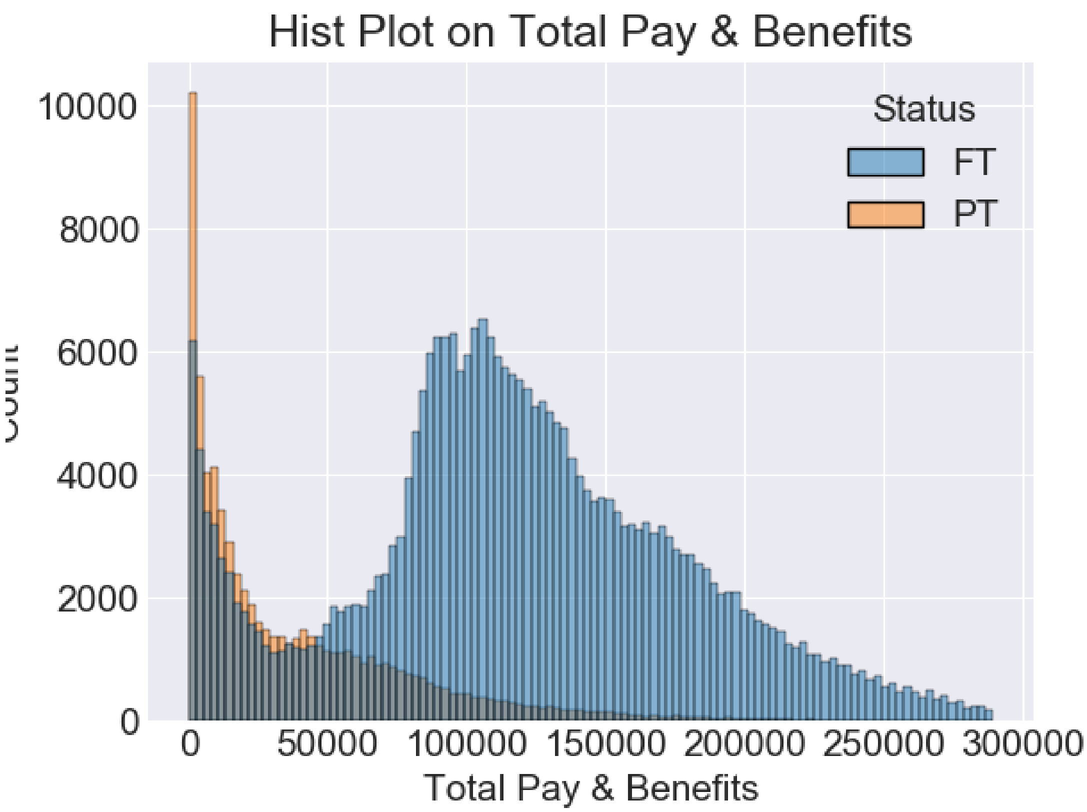
Pair Plot

This shows the linear relationship between the numeric features. A positive linear relationship can be observed more than negative linear relationship.



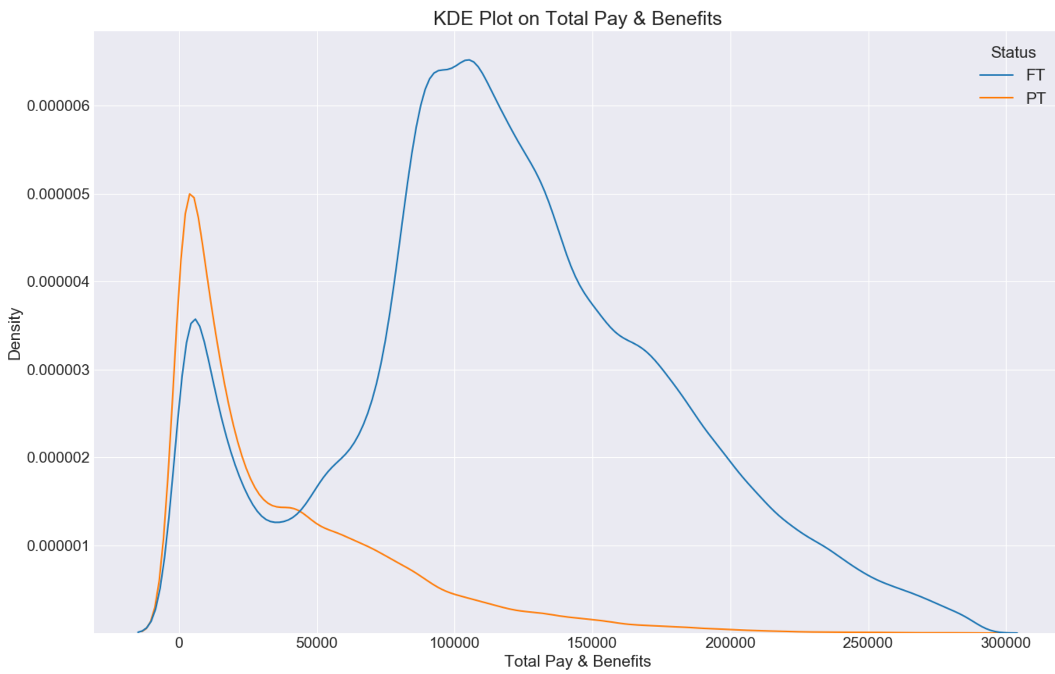
Hist Plot

The hist plot still indicate same observations as stated in the distribution plot above.



KDE Plot

This plot indicates two modal class for Total Pay & Benefits for FT workers while a single modal class for PT workers.

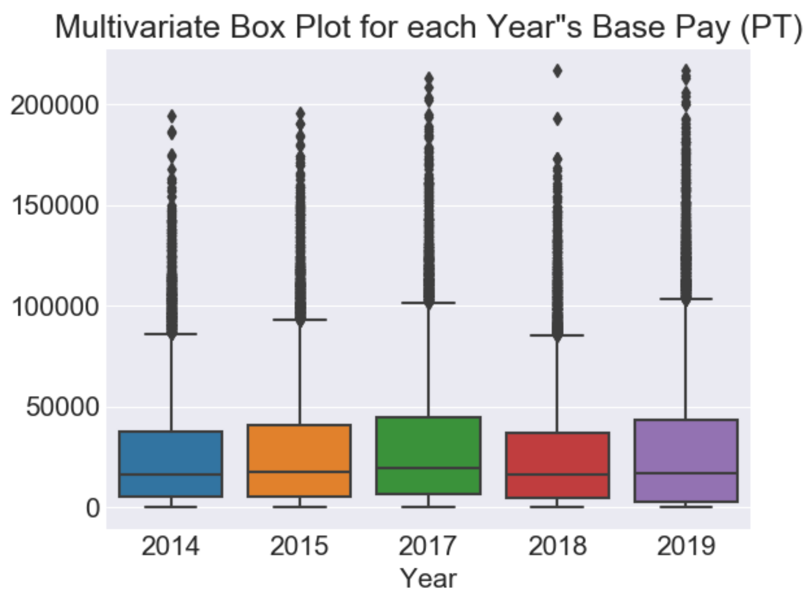


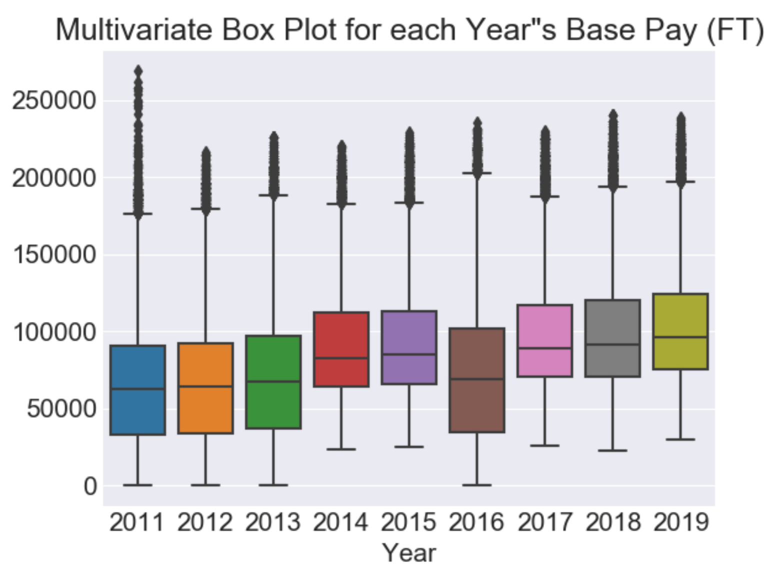
Regression Plot

The regression plot between Base Pay and dependent variable shows a very positive trend and relationship that as Base Pay Increases Total Pay and Benefits increases too vice versa.

Multivariate Box Plot

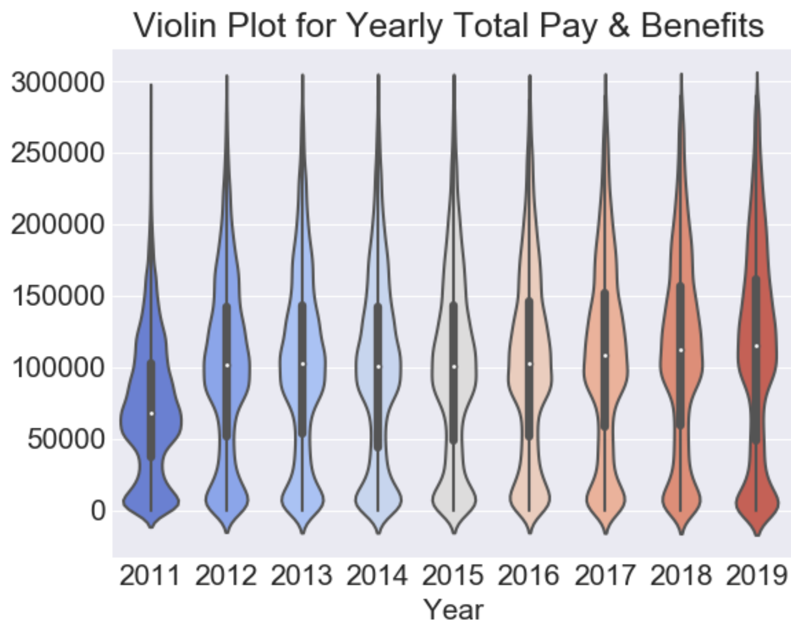
The boxplot shows us the percentile values at different juncture of 25% interval and likewise the distribution of Base Pay for PT and FT workers over the years.



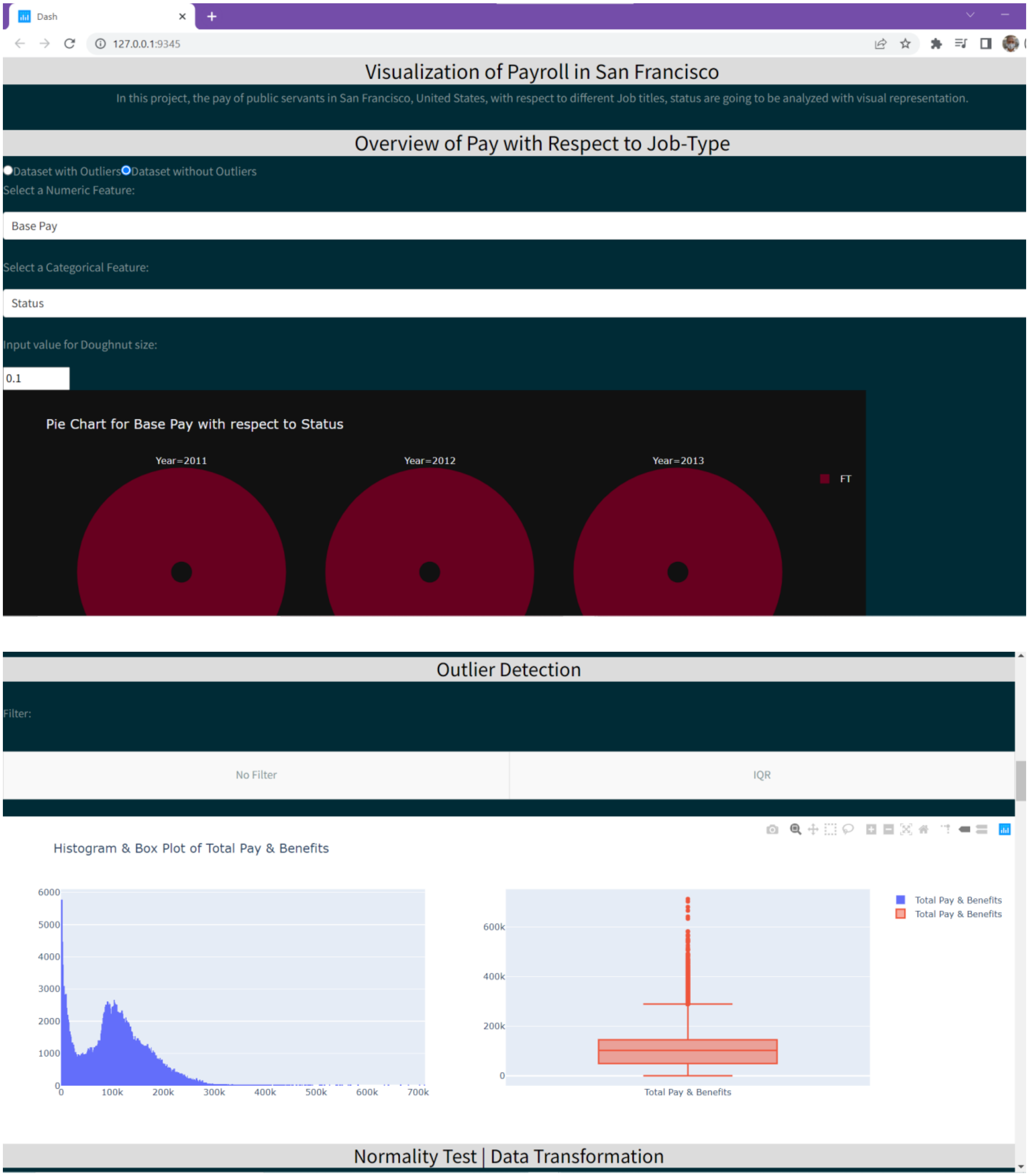


Violin Plot

We can observe the distribution of Total Pay & Benefits over the years (2011-2019) and also notice an upward trend over the years.



Dash App



Dash
127.0.0.1:9345
Visualization of Payroll in San Francisco
In this project, the pay of public servants in San Francisco, United States, with respect to different Job titles, status are going to be analyzed with visual representation.
Overview of Pay with Respect to Job-Type
Dataset with Outliers
Dataset without Outliers
Select a Numeric Feature:
Base Pay
Select a Categorical Feature:
Status
Input value for Doughnut size:
0.1
Pie Chart for Base Pay with respect to Status
Year=2011
Year=2012
Year=2013
FT
Outlier Detection
Filter:
No Filter
IQR
Histogram & Box Plot of Total Pay & Benefits
Total Pay & Benefits
Total Pay & Benefits
Total Pay & Benefits
Normality Test | Data Transformation

Normality Test | Data Transformation

- Dataset with Outliers
- Dataset without Outliers

Transformation Type:

- Quantile Transformation
- Box-Cox
- Reciprocal Transform
- Square Root Transform

Select Testing Procedure:

Shapiro"s Test

Shapiro test: statistics = 0.97 p-vlaue of =0.00 Total Pay & Benefits dataset is NOT Normal

Statistics

Features	mean	median
Base Pay	69818.20211859654	68326.5
Overtime Pay	5657.256193132644	0
Other Pay	3355.513903255947	704.7
Benefits	22534.488563568273	27190.72

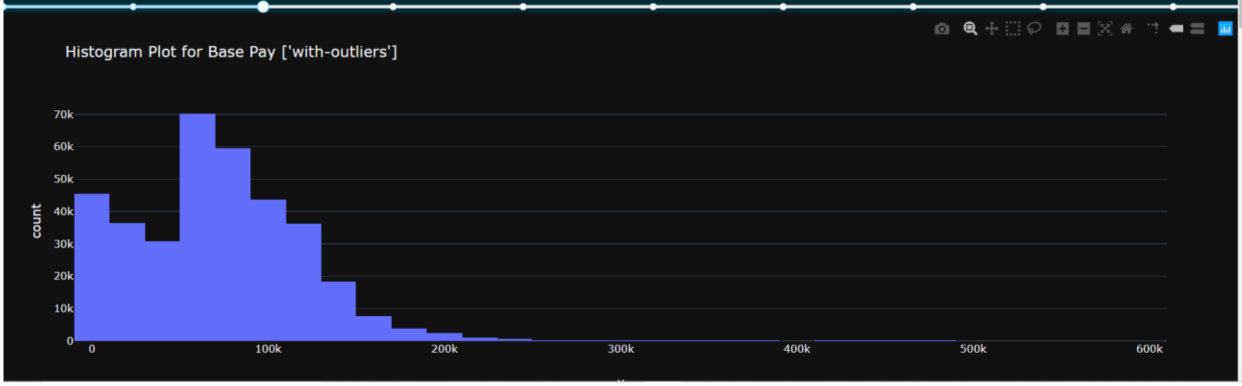
Visualizations

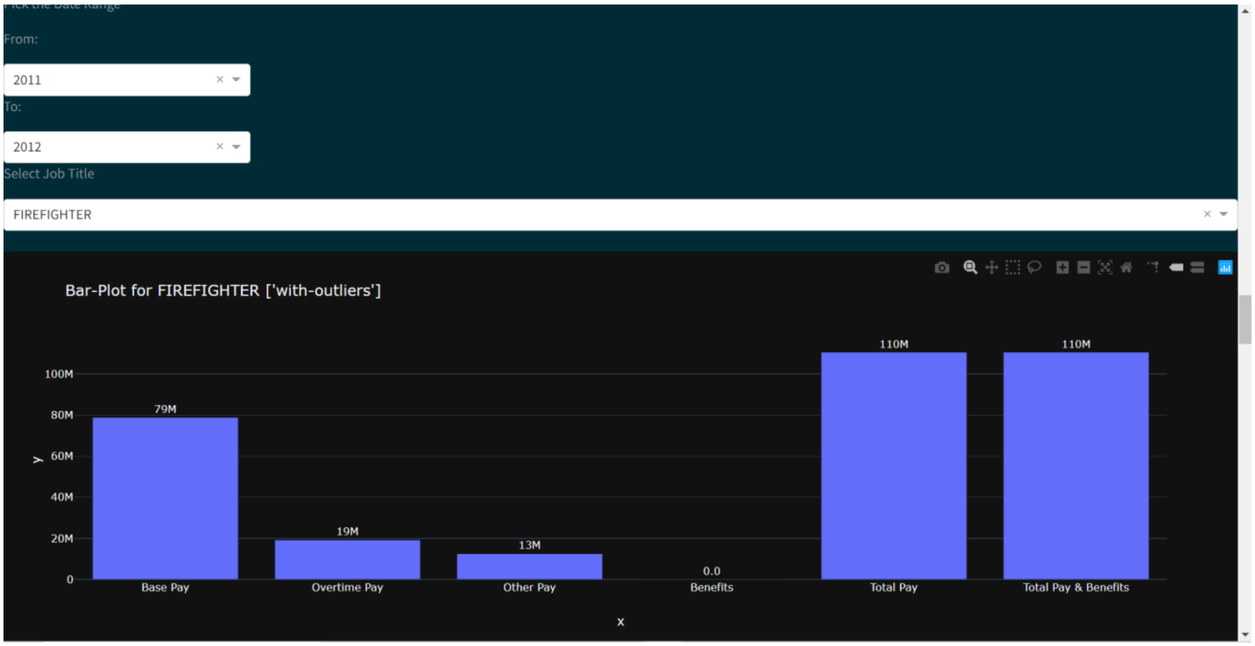
Histogram Plot

- Dataset with Outliers
- Dataset without Outliers

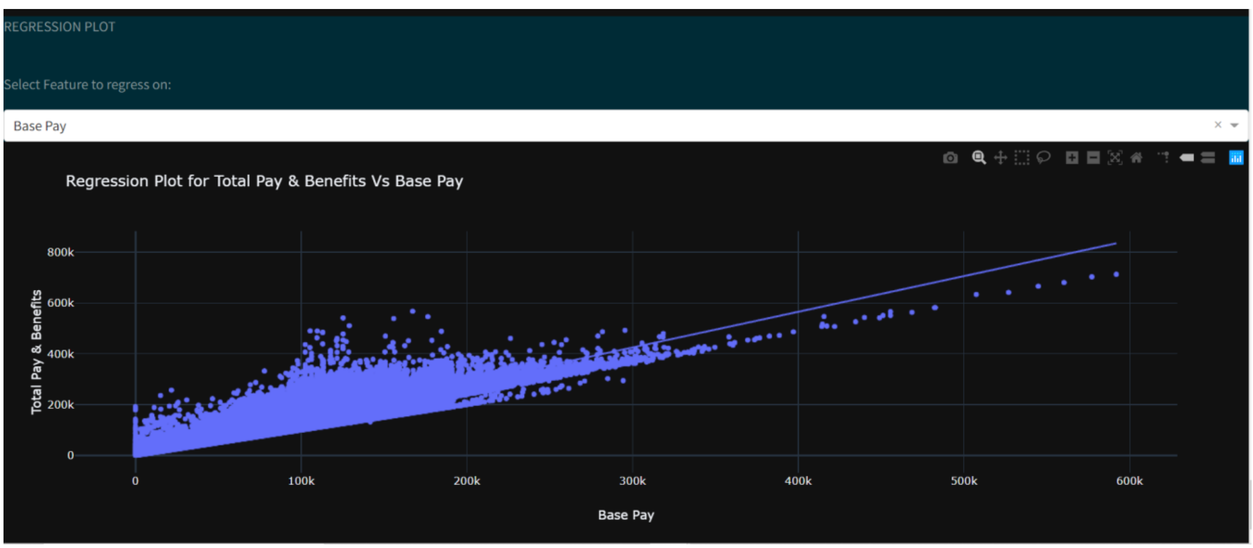
Select a Numeric Feature:

Base Pay





From:
2011
To:
2012
Select Job Title
FIREFIGHTER
Bar-Plot for FIREFIGHTER ['with-outliers']
100M
80M
60M
40M
20M
0
y
79M
19M
13M
0.0
110M
110M
Base Pay
Overtime Pay
Other Pay
Benefits
Total Pay
Total Pay & Benefits
x



REGRESSION PLOT
Select Feature to regress on:
Base Pay
Regression Plot for Total Pay & Benefits Vs Base Pay
800k
600k
400k
200k
0
Total Pay & Benefits
0
100k
200k
300k
400k
500k
600k
Base Pay

Summary & Recommendations

With the various generated plots for the San Francisco payroll between 2011-2019, I can infer that:

- Full time workers are more likely to get a job in San Francisco.
- The typical base pay for part-time workers is less than $50,000.
- The typical base pay for full-time workers is between than $100,000 and $150,000.
- There exist two categories of full-time workers, one group paid as part-time. workers and the other group as actual full-time workers even though all exist in full-time status.

The charts tell similar and different information to different individuals with its discreteness.

The created app with dash also transfers same insights from the charts generated with plotly. The app is very interactive being that you can change values and features and observe pre-defined changes and trends.

References

https://pyshark.com/test-for-normality-using-python/

https://machinelearningmastery.com/quantile-transforms-for-machine-learning/

https://stackoverflow.com/questions/25039626/how-do-i-find-numeric-columns-in-pandas

APPENDIX

```
print("#============Osemekhian Ehilen DV Project======================")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy
import os
from tabulate import tabulate
import statsmodels.api as sm
from scipy import signal
from scipy.stats import shapiro
from scipy.stats import kstest
from scipy.stats import normaltest
from scipy import stats
import plotly.express as px
import plotly.graph_objs as go
from plotly.subplots import make_subplots
plt.style.use('seaborn-darkgrid')
font={'family':'serif','color':'black','size':12}

#============Helper Function===============

def shapiro_test(x, title):
    stats, p = shapiro(x)
    print('=' * 50)
    print(f'Shapiro test : {title} dataset : statistics = {stats:.2f} p-
vlaue of ={p:.2f}' )
    alpha = 0.01
    if p > alpha :
        print(f'Shapiro test: {title} dataset is Normal')
    else:
        print(f'Shapiro test: {title} dataset is NOT Normal')
    print('=' * 50)
```

```
def ks_test(x, title):
    mean = np.mean(x)
    std = np.std(x)
    dist = np.random.normal(mean, std, len(x))
    stats, p = kstest(x, dist)
    print('='*50)
    print(f'K-S test: {title} dataset: statistics= {stats:.2f} p-value =
{p:.2f}' )

    alpha = 0.01
    if p > alpha :
        print(f'K-S test:  {title} dataset is Normal')
    else:
        print(f'K-S test : {title} dataset is Not Normal')
    print('=' * 50)


def da_k_squared_test(x, title):
    stats, p = normaltest(x)
    print('='*50)
    print(f'da_k_squared test: {title} dataset: statistics= {stats:.2f} p-
value = {p:.2f}' )

    alpha = 0.01
    if p > alpha :
        print(f'da_k_squaredtest:  {title} dataset is Normal')
    else:
        print(f'da_k_squared test : {title} dataset is Not Normal')
    print('=' * 50)

#=========================================

#path="C:/Users/oseme/Desktop/Data Visualization Class/Project/"
df= pd.read_csv("san-francisco-payroll_2011-2019.csv",low_memory=False)
print(df.head())
print(df.info())
print(df.isna().sum())

#==cleaning
df['Status'].fillna(value=df['Status'].mode()[0],inplace=True)
df= df[~(df["Base Pay"]=="Not Provided")]  #Remove Base pay rows with Not
Provided
df['Benefits'][df.Benefits=="Not Provided"]=0
df['Overtime Pay'][df["Overtime Pay"]=="Not Provided"]=0
df['Other Pay'][df["Other Pay"]=="Not Provided"]=0
df['Status']=df['Status'].map(lambda x:1 if x== 'FT' else 0)
df=df[~(df['Total Pay & Benefits']==0)]
# change data type
print(df.iloc[:,2:6])
df_clean=df.astype(dict(zip(df.columns[2:6],[float]*4)))
#== Reverse dataframe
df_clean=df_clean.iloc[::-1,:].reset_index().drop(columns=["index"])
```

```
#== Make pay positive where negative
df_clean['Total Pay & Benefits']=np.abs(df_clean['Total Pay & Benefits'])
df_clean['Base Pay']=np.abs(df_clean['Base Pay'])
df_clean['Overtime Pay']=np.abs(df_clean['Overtime Pay'])
df_clean['Other Pay']=np.abs(df_clean['Other Pay'])
df_clean['Total Pay']=np.abs(df_clean['Total Pay'])
df_clean['Benefits']=np.abs(df_clean['Benefits'])

print(df_clean.info())
# print(tabulate(df_clean.info(),headers='keys',tablefmt="fancy_grid"))
# df_clean.to_excel("clean_df.xlsx")
print("#============Outlier Detection======================")
# Outlier Detection
plt.figure()
plt.hist(df['Total Pay & Benefits'],bins=50)
plt.grid()
plt.xlabel('Total Pay & Benefits')
plt.ylabel('Magnitude')
plt.title('Histogram for \nTotal Pay & Benefits',fontdict=font)
plt.show()

#== Boxplot
plt.figure()
plt.boxplot(df['Total Pay & Benefits'])
plt.grid()
plt.xlabel('Total Pay & Benefits')
plt.ylabel('USD($)')
plt.title('Boxplot for \nTotal Pay & Benefits',fontdict=font)
plt.show()


def outlier(data):
 global Q1,Q3
 sorted(data)
 Q1,Q3 = np.percentile(data , [25,75])
 IQR = Q3-Q1
 lower_range = Q1 - (1.5 * IQR)
 upper_range = Q3 + (1.5 * IQR)
 return lower_range,upper_range

lower,upper= outlier(df_clean['Total Pay & Benefits'])
df_no_outlier= df_clean[(df_clean['Total Pay & Benefits']<upper)]
print(tabulate(pd.DataFrame(df_no_outlier.describe().iloc[:,-
3]),headers='keys',tablefmt="fancy_grid"))
#== Boxplot
plt.figure()
plt.boxplot(df_no_outlier['Total Pay & Benefits'])
plt.grid()
plt.xlabel('Total Pay & Benefits')
plt.ylabel('USD($)')
plt.title('Boxplot for \nTotal Pay & Benefits',fontdict=font)
plt.show()

#== Histogram
```

```
plt.figure()
plt.hist(df_no_outlier['Total Pay & Benefits'],bins=50)
plt.grid()
plt.xlabel('Total Pay & Benefits')
plt.ylabel('Magnitude')
plt.title('Histogram for \nTotal Pay & Benefits',fontdict=font)
plt.show()

print("#============SVD| Condition Number | PCA======================")
# SVD| Condition Number | PCA

#Scale features
from sklearn.preprocessing import StandardScaler
x=df_no_outlier[['Base Pay','Overtime Pay','Other Pay','Other
Pay','Benefits','Status']]
sc=StandardScaler()
data_scaled= sc.fit_transform(x)

# SVD
H= np.matmul(x.values.T,x.values)
#SVD
_,d,_=np.linalg.svd(H)
res=pd.DataFrame(d,index=x.columns, columns=['Singular Values'])
# print(res)
print(tabulate(res,headers='keys',tablefmt="fancy_grid"))
# print(f'Condition number for X Features is {np.linalg.cond(x)}')

cond1=np.linalg.cond(x)
condition=pd.DataFrame(data=[cond1],columns=['Condition Number'])
print(tabulate(condition,headers='keys',tablefmt="fancy_grid"))



from sklearn.decomposition import PCA
pca=PCA(n_components='mle',svd_solver='full')
transformed=pca.fit_transform(data_scaled)
print(f'Explained Variance: \n {pca.explained_variance_ratio_}')

# Plot explained variance
plt.figure()
x=np.arange(1,len(pca.explained_variance_ratio_)+1)
plt.xticks(x)
plt.plot(x,np.cumsum(pca.explained_variance_ratio_),c='red',marker='*')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance vs Number of Components')
plt.show()

# Transformed Data From PCA
xnew=transformed.copy()
H= np.matmul(xnew.T,xnew)
#SVD new
_,d,_=np.linalg.svd(H)
res=pd.DataFrame(d, columns=['Singular Values'])
```

```
# print(res)
print(tabulate(res,headers='keys',tablefmt="fancy_grid"))
# print(f'Condition number for Reduced Features is
{np.linalg.cond(xnew)}')
cond2=np.linalg.cond(xnew)
condition=pd.DataFrame(data=[cond2],columns=['Condition Number'])
print(tabulate(condition,headers='keys',tablefmt="fancy_grid"))
# Obviously Status is the not required feature with almost 0 singular
value

print("#============Normality Test======================")
# Normality Test
#perform Shapiro-Wilk test for normality
print(shapiro(df_no_outlier['Total Pay & Benefits']))
print(shapiro_test(df_no_outlier['Total Pay & Benefits'],'Total Pay &
Benefits'))
print(ks_test(df_no_outlier['Total Pay & Benefits'],'Total Pay &
Benefits'))
print(da_k_squared_test(df_no_outlier['Total Pay & Benefits'],'Total Pay &
Benefits'))

#== qqplot without Outliers
sm.qqplot(df_no_outlier['Total Pay & Benefits'], line ='s')
plt.title('qqplot',fontdict=font)
plt.show()

# target_trans = stats.norm.ppf(stats.rankdata(df_no_outlier['Total Pay &
Benefits'])/(len(df_no_outlier['Total Pay & Benefits']) + 1))

print("#============Transformation Using Quantile
Transformer======================")
#============Transformation Using Quantile
Transformer======================

from sklearn.preprocessing import QuantileTransformer
quantile = QuantileTransformer(output_distribution='normal')
data_trans = quantile.fit_transform(df_no_outlier['Total Pay &
Benefits'].values.reshape(-1,1))
print('=====Check Transformed Normality=========')
print(shapiro(data_trans))
# check
plt.figure()
plt.hist(data_trans)
plt.title('Quantile Transformation')
plt.ylabel("Magnitude")
plt.show()

#stat check
print(shapiro_test(data_trans.ravel(),'Total Pay & Benefits'))
print(ks_test(data_trans.ravel(),'Total Pay & Benefits'))
print(da_k_squared_test(data_trans.ravel(),'Total Pay & Benefits'))
# quantile.inverse_transform(np.array([[0.8]]))

print("#============Heatmap & Pearson Correlation Coefficient
```

```
Matrix======================")
corr= df_no_outlier.select_dtypes(include='float64')
corr= corr.corr()
# Heatmap
sns.heatmap(corr,annot=True)
plt.title(f"Pearson Correlation Coefficient")
plt.show()
#Corr Matrix
pd.plotting.scatter_matrix(df_no_outlier.select_dtypes(include='float64'),
                           hist_kwds={'bins':50},alpha=0.5)
plt.suptitle(f'Correlation Matrix ')
plt.rcParams.update({'font.size': 22})
plt.show()

print("#============Statistics======================")
df_no_outlier['Status']=df_no_outlier['Status'].map(lambda x:'FT' if x== 1
else 'PT')
features=['mean','median']
dfc= df_no_outlier.select_dtypes(include='float64')
cols=dfc.columns
stat_df= pd.DataFrame(columns=features,index=cols)
stat_df.loc[cols[0]]=[dfc[cols[0]].mean(),dfc[cols[0]].median()]
stat_df.loc[cols[1]]=[dfc[cols[1]].mean(),dfc[cols[1]].median()]
stat_df.loc[cols[2]]=[dfc[cols[2]].mean(),dfc[cols[2]].median()]
stat_df.loc[cols[3]]=[dfc[cols[3]].mean(),dfc[cols[3]].median()]
stat_df.loc[cols[4]]=[dfc[cols[4]].mean(),dfc[cols[4]].median()]
stat_df.loc[cols[5]]=[dfc[cols[5]].mean(),dfc[cols[5]].median()]
stat_df=stat_df.round(2)
stat_df.index.name= "Statistics"
print(tabulate(stat_df,headers='keys',tablefmt="fancy_grid"))

print("#============Visualization With Seaborn======================")

#Line Plots
target= df_no_outlier['Total Pay & Benefits']
plt.figure(figsize=(10,8))
sns.lineplot(data=df_no_outlier[df_no_outlier['Status']=='FT'],x='Year',y=
'Total Pay & Benefits',hue='Status')
plt.rcParams.update({'font.size': 22})
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("Trend of Total Pay & Benefit \nfor PartTime Employees")
plt.show()

plt.figure(figsize=(10,8))
sns.lineplot(data=df_no_outlier[df_no_outlier['Status']=='PT'],x='Year',y=
'Total Pay & Benefits',hue='Status')
plt.rcParams.update({'font.size': 22})
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("Trend of Total Pay & Benefit \nfor FullTime Employees")
plt.show()

#Count Plot
```

```python
dff=df_no_outlier.copy()
dff.Year=dff.Year.astype('str')
plt.figure(figsize=(9,7))
sns.countplot(data=dff,x='Year',hue='Status')
plt.title("Count of Job Type")
plt.ylabel('Count')
plt.rcParams.update({'font.size':15 })
plt.show()

#Bar plot
sns.barplot(x='Year', y='Total Pay & Benefits', hue="Status",
data=df_no_outlier)
plt.xlabel('Year')
plt.ylabel('Total Pay & Benefits')
plt.title('Bar Plot of Yearly Total Pay & Benefits')
plt.show()

#Cat Plot
plt.figure(figsize=(10,8))
sns.catplot(data=df_no_outlier, x="Total Pay & Benefits", y="Status",
            kind='box')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title('BoxPlot on Total Pay & Benefits',fontdict=font)
plt.show()

# Pie plot

def func(pct, allvals):
    absolute = int(round(pct / 100. * np.sum(allvals)))
    return "{:d}".format(absolute)
dff=df_no_outlier.copy()
dff['Status']=dff['Status'].map(lambda x:1 if x== 'FT' else 0)

#Pie chart single
plt.figure()
plt.pie(dff['Status'].value_counts(),labels=["Full Time","Part Time"],
explode=[0,0.05],autopct='%1.2f%%')
plt.title("Pie Chart of Job Status (2011-2019)")
plt.legend(loc=(0.8,0.8))
plt.axis('square')
plt.show()

# Dashboard PiePlots
fig, ax = plt.subplots(3,3, figsize=(18,10))
ax[0,0].pie(dff[dff.Year==2011].Status.value_counts(),labels=["Full
Time"],autopct='%1.2f%%')
ax[0,0].set_title("Pie Chart of Job Status (2011)")
ax[0,0].legend(loc=(0.8,0.8))

ax[0,1].pie(dff[dff.Year==2012].Status.value_counts(),labels=["Full
Time"],autopct='%1.2f%%')
ax[0,1].set_title("Pie Chart of Job Status (2012)")
ax[0,1].legend(loc=(0.8,0.8))
```

```
ax[0,2].pie(dff[dff.Year==2013].Status.value_counts(),labels=["Full
Time"],autopct='%1.2f%%')
ax[0,2].set_title("Pie Chart of Job Status (2013)")
ax[0,2].legend(loc=(0.8,0.8))

ax[1,0].pie(dff[dff.Year==2014].Status.value_counts(),labels=["Full
Time","Part Time"],explode=[0,0.05],autopct='%1.2f%%')
ax[1,0].set_title("Pie Chart of Job Status (2014)")
ax[1,0].legend(loc=(0.8,0.8))

ax[1,1].pie(dff[dff.Year==2015].Status.value_counts(),labels=["Full
Time","Part Time"],explode=[0,0.05],autopct='%1.2f%%')
ax[1,1].set_title("Pie Chart of Job Status (2015)")
ax[1,1].legend(loc=(0.8,0.8))

ax[1,2].pie(dff[dff.Year==2016].Status.value_counts(),autopct='%1.2f%%')
ax[1,2].set_title("Pie Chart of Job Status (2016)")
ax[1,2].legend(loc=(0.8,0.8))

ax[2,0].pie(dff[dff.Year==2017].Status.value_counts(),labels=["Full
Time","Part Time"],explode=[0,0.05],autopct='%1.2f%%')
ax[2,0].set_title("Pie Chart of Job Status (2017)")
ax[2,0].legend(loc=(0.8,0.8))

ax[2,1].pie(dff[dff.Year==2018].Status.value_counts(),labels=["Full
Time","Part Time"],explode=[0,0.05],autopct='%1.2f%%')
ax[2,1].set_title("Pie Chart of Job Status (2018)")
ax[2,1].legend(loc=(0.8,0.8))

ax[2,2].pie(dff[dff.Year==2019].Status.value_counts(),labels=["Full
Time","Part Time"],explode=[0,0.05],autopct='%1.2f%%')
ax[2,2].set_title("Pie Chart of Job Status (2019)")
ax[2,2].legend(loc=(0.8,0.8))

plt.tight_layout()
plt.show()

#Recurrent Jobs
recur= dff['Job Title'].value_counts()[dff['Job
Title'].value_counts()>5000]
plt.figure(figsize=(16,13))
plt.pie(recur,labels=recur.index,autopct='%1.2f%%')
plt.title("Pie Chart for Recurrent Jobs Over 5000 (2011-2019)")
plt.show()


#Displot
sns.displot(data=df_no_outlier, x="Total Pay & Benefits",col='Status')
plt.show()

# Pair Plot
sns.pairplot(df_no_outlier.select_dtypes(include=['float64']))
plt.show()
```

```
#Hist Plot
sns.histplot(data=df_no_outlier, x="Total Pay & Benefits", hue='Status')
plt.title("Hist Plot on Total Pay & Benefits")
plt.show()

#KDE Plot
plt.figure(figsize=(16,10))
sns.kdeplot(data=df_no_outlier, x="Total Pay & Benefits", hue='Status')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("KDE Plot on Total Pay & Benefits")
plt.show()

# Lmplot
plt.figure(figsize=(18,10))
sns.lmplot(data=df_no_outlier, x="Base Pay", y="Total Pay & Benefits")
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
#plt.title("LM-Plot | Total Pay & Benefits vs Base Pay")
plt.show()

#Multivariate Box plot
sns.boxplot(x='Year',y='Base
Pay',data=df_no_outlier[df_no_outlier.Status=='FT'])
plt.title('Multivariate Box Plot for each Year"s Base Pay (FT)')
plt.show()

sns.boxplot(x='Year',y='Base
Pay',data=df_no_outlier[df_no_outlier.Status=='PT'])
plt.title('Multivariate Box Plot for each Year"s Base Pay (PT)')
plt.show()

# Violoin Plot
sns.violinplot(x="Year", y="Total Pay & Benefits", data=df_no_outlier,
palette="coolwarm")
plt.title('Violin Plot for Yearly Total Pay & Benefits ')
plt.show()




#References
#https://pyshark.com/test-for-normality-using-python/
#https://machinelearningmastery.com/quantile-transforms-for-machine-
learning/
#https://stackoverflow.com/questions/25039626/how-do-i-find-numeric-
columns-in-pandas
```

APPENDIX 2

```
#============================================
import random
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objs as go
from plotly.subplots import make_subplots
from chart_studio import plotly
import scipy
import os
from tabulate import tabulate
import statsmodels.api as sm
from statsmodels.graphics.gofplots import qqplot
from scipy.stats import shapiro
from scipy import stats
from sklearn.preprocessing import QuantileTransformer

import dash as dash
from dash import dcc, dash_table
from dash import html
from dash.dependencies import Input, Output
from dash.exceptions import PreventUpdate
import dash_daq as daq
import dash_bootstrap_components as dbc
import warnings
from scipy import signal
import numpy as np
from scipy.stats import shapiro
from scipy.stats import kstest
from scipy.stats import normaltest
from datetime import date
warnings.filterwarnings('ignore')

style={'textAlign':'center','background': 'rgb(220, 220, 220)','color':
'black'}
style2={'textAlign':'center'}
steps=0.1
marks= lambda min,max:{i:f"{i}" for i in range(min,max)}
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
BS=
["https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
]

#============Helper Function===============
def qqp(values):
    qqplot_data = qqplot(values, line='s').gca().lines
    fig = go.Figure()

    fig.add_trace({
        'type': 'scatter',
        'x': qqplot_data[0].get_xdata(),
        'y': qqplot_data[0].get_ydata()})
```

```python
    fig.add_trace({
        'type': 'scatter',
        'x': qqplot_data[1].get_xdata(),
        'y': qqplot_data[1].get_ydata(),
        'mode': 'lines'})
    fig['layout'].update({
        'title': 'Quantile-Quantile Plot',
        'xaxis': {
            'title': 'Theoritical Quantities',
            'zeroline': False
        },
        'yaxis': {
            'title': 'Sample Quantities'
        },
        'showlegend': False,
        'width': 800,
        'height': 700,
    })
    return fig

def shapiro_test(x, title):
    stats, p = shapiro(x)
    alpha = 0.01
    if p > alpha :
        return f'Shapiro test:\n statistics = {stats:.2f} p-vlaue of
={p:.2f} \n{title} dataset is Normal'
    else:
        return f'Shapiro test:\n statistics = {stats:.2f} p-vlaue of
={p:.2f} \n {title} dataset is NOT Normal'


def ks_test(x, title):
    mean = np.mean(x)
    std = np.std(x)
    dist = np.random.normal(mean, std, len(x))
    stats, p = kstest(x, dist)

    alpha = 0.01
    if p > alpha :
        return f'K-S test:\n statistics = {stats:.2f} p-vlaue of ={p:.2f}
\n  {title} dataset is Normal'
    else:
        return f'K-S test :\n statistics = {stats:.2f} p-vlaue of ={p:.2f}
\n {title} dataset is Not Normal'


def da_k_squared_test(x, title):
    stats, p = normaltest(x)
    alpha = 0.01
    if p > alpha :
        return f'da_k_squaredtest:\n statistics = {stats:.2f} p-vlaue of
={p:.2f} \n  {title} dataset is Normal'
    else:
        return f'da_k_squared test :\n statistics = {stats:.2f} p-vlaue of
```

```
={p:.2f} \n {title} dataset is Not Normal'

#===========================================


intro= """ In this project, the pay of public servants in San Francisco,\n
United States,\n with respect to different Job titles,\n
status are going to be analyzed with visual representation.
 """
#=====================================================
#path="C:/Users/oseme/Desktop/Data Visualization Class/Project/"
df= pd.read_csv("san-francisco-payroll_2011-2019.csv",low_memory=False)

#==cleaning
df['Status'].fillna(value=df['Status'].mode()[0],inplace=True)
df= df[~(df["Base Pay"]=="Not Provided")]  #Remove Base pay rows with Not
Provided
df['Benefits'][df.Benefits=="Not Provided"]=0
df['Overtime Pay'][df["Overtime Pay"]=="Not Provided"]=0
df['Other Pay'][df["Other Pay"]=="Not Provided"]=0
df['Status']=df['Status'].map(lambda x:1 if x== 'FT' else 0)
df=df[~(df['Total Pay & Benefits']==0)]
# change data type
# print(df.iloc[:,2:6])
df_clean=df.astype(dict(zip(df.columns[2:6],[float]*4)))
#== Reverse dataframe
df_clean=df_clean.iloc[::-1,:].reset_index().drop(columns=["index"])

#== Make pay positive where negative
df_clean['Total Pay & Benefits']=np.abs(df_clean['Total Pay & Benefits'])
df_clean['Base Pay']=np.abs(df_clean['Base Pay'])
df_clean['Overtime Pay']=np.abs(df_clean['Overtime Pay'])
df_clean['Other Pay']=np.abs(df_clean['Other Pay'])
df_clean['Total Pay']=np.abs(df_clean['Total Pay'])
df_clean['Benefits']=np.abs(df_clean['Benefits'])

print(df_clean.info())
def outlier(data):
 global Q1,Q3
 sorted(data)
 Q1,Q3 = np.percentile(data , [25,75])
 IQR = Q3-Q1
 lower_range = Q1 - (1.5 * IQR)
 upper_range = Q3 + (1.5 * IQR)
 return lower_range,upper_range

lower,upper= outlier(df_clean['Total Pay & Benefits'])
df_no_outlier= df_clean[(df_clean['Total Pay & Benefits']<upper)]
print(tabulate(pd.DataFrame(df_no_outlier.describe().iloc[:,-
3]),headers='keys',tablefmt="fancy_grid"))

print("#===========Transformation Using Quantile
Transformer=====================")
#===========Transformation Using Quantile
```

```
Transformer=====================

quantile = QuantileTransformer(output_distribution='normal')
data_trans = quantile.fit_transform(df_no_outlier['Total Pay &
Benefits'].values.reshape(-1,1))

print("#===========Statistics=====================")
df_no_outlier['Status']=df_no_outlier['Status'].map(lambda x:'FT' if x== 1
else 'PT')
features=['mean','median']
dfc= df_no_outlier.select_dtypes(include='float64')
cols=dfc.columns
stat_df= pd.DataFrame(columns=features,index=cols)
stat_df.loc[cols[0]]=[dfc[cols[0]].mean(),dfc[cols[0]].median()]
stat_df.loc[cols[1]]=[dfc[cols[1]].mean(),dfc[cols[1]].median()]
stat_df.loc[cols[2]]=[dfc[cols[2]].mean(),dfc[cols[2]].median()]
stat_df.loc[cols[3]]=[dfc[cols[3]].mean(),dfc[cols[3]].median()]
stat_df.loc[cols[4]]=[dfc[cols[4]].mean(),dfc[cols[4]].median()]
stat_df.loc[cols[5]]=[dfc[cols[5]].mean(),dfc[cols[5]].median()]
stat_df=stat_df.round(2)
stat_df.index.name= "Statistics"
#print(tabulate(stat_df,headers='keys',tablefmt="fancy_grid"))
stat_df= stat_df.round({'mean':2,'median':2})
stat_df.insert(0,'Features',list(stat_df.index))
statistics= stat_df

cat= ['Job Title','Status']
cat2= ['Base Pay', 'Overtime Pay', 'Other Pay','Benefits', 'Total Pay',
'Total Pay & Benefits']
df_clean['Status']=df_clean['Status'].map(lambda x:'FT' if x== 1 else
'PT')

countdf= df_clean.groupby('Year').count()
figcount=px.bar(countdf, y='Status', title="    Employee Count Over The
Years ",
                      template="plotly_dark",text='Status')
figcount.update_traces(texttemplate='%{text:.2s}', textposition='outside')
figcount.update_layout(uniformtext_minsize=8, uniformtext_mode='hide')









#============================================

main_df = df_no_outlier

my_app= dash.Dash(__name__,external_stylesheets=[dbc.themes.SOLAR])
#dbc.themes.MORPH
```

```
my_app.layout= html.Div(children=[
    html.H3("Visualization of  Payroll in San Francisco",style=style),
    html.H6(intro,style=style2),
    html.Br(),


    #Pie Seciton
    html.H3("Overview of Pay with Respect to Job-Type",style=style),
    dcc.RadioItems(id='overradio',
                                   options=[{"label": "Dataset with
Outliers", "value": "with-outliers"},
                                            {"label": "Dataset without
Outliers", "value": "without-outliers"}],
                                   value='without-outliers',
labelStyle={'display': 'in-line'}),
    html.P("Select a Numeric Feature:"),
    dcc.Dropdown(id='overcheck',
                 options=[{'label':i,'value':i} for i in cat2],
                 value='Base Pay',placeholder='Select one...'),html.Br(),
    html.P("Select a Categorical Feature:"),
    dcc.Dropdown(id='overcheck2',
                 options=[{'label': i, 'value': i} for i in cat],
                 value='Status',placeholder='Select one...'), html.Br(),
    html.P("Input value for Doughnut size:"),
    dcc.Input(id='overin',type='number',value=0.1,min=0.01, max=0.9,
step=0.01),

html.Div(id='overview',style={'width':'80%','height':'80%','horizontal-
align': 'middle'}),html.Br(),
    #Pie Section Ends

    html.Br(),
    #Outlier Section
    html.H3("Outlier Detection",style=style), html.Br(),
    html.Div([
                html.P("Filter:"),html.Br(),
                dcc.Tabs(id='filter',children=[
                    dcc.Tab(label='No Filter',value='No filter'),
                    dcc.Tab(label='IQR',value='IQR')
                ]),
                html.Br(),
                dcc.Graph(id='g1')
    ],id='outlier'),
    #Outlier End

    #Normality section
    html.H3("Normality Test | Data Transformation",style=style),
html.Br(),
    html.Div([
```

```
                dcc.RadioItems(id='normradio',
                                   options=[{"label": "Dataset with
Outliers", "value": "with-outliers"},
                                            {"label": "Dataset without
Outliers", "value": "without-outliers"}],
                                   value='with-outliers',
labelStyle={'display': 'block'}),
                    html.Br(),
                html.P("Transformation Type:"),
                dcc.RadioItems(id='transform',
                                   options=[{"label": "Quantile
Transformation", "value": "quantile"},
                                            {"label": "Box-Cox", "value":
"box"},
                                            {"label": "Reciprocal
Transform", "value": "reciprocal"},
                                            {"label": "Square Root
Transform", "value": "sqrt"}],
                                   value='with-outliers',
labelStyle={'display': 'block'}),html.Br(),
                    html.P("Select Testing Procedure:"),
                    dcc.Dropdown(id='normdrop',options=[
                                    {'label': 'Shapiro"s Test', 'value':
'shapiro'},
                                    {'label': 'Kolmogorov-Smirnov Test',
'value': 'ks'},
                                    {'label': 'D"Agostino-Pearson Test',
'value': 'dap'},
                                    {'label': 'Histogram', 'value':
'hist'},
                                    {'label': 'QQ-Plot', 'value': 'qq'}
                                ],value='shapiro',clearable=False),
                    html.Br(),
                    html.Div(id='normal-out')
    ],id='normality'), html.Br(),
    #Normality End
    html.Br(),
    html.H3("Statistics",style=style), html.Br(),
    html.Div([
        html.Div(dash_table.DataTable(statistics.to_dict('records'),
                                      [{"name": i, "id": i} for i in
statistics.columns],
                                      style_cell={'textAlign':
'left','padding':'5px'},
                                      style_header={'backgroundColor':
'rgb(220, 220, 220)','fontWeight': 'bold'}))
    ],id='statistics'), html.Br(),

    #Viz Start
    html.H3("Visualizations",style=style), html.Br(),
    html.P('Histogram Plot'),
    html.Div(children=[
                        dcc.RadioItems(id='vizradio',
                                   options=[{"label": "Dataset with
```

```
Outliers", "value": "with-outliers"},
                                             {"label": "Dataset without
Outliers", "value": "without-outliers"}],
                                    value='with-outliers',
labelStyle={'display': 'block'}),
                        html.Br(),
                        html.P("Select a Numeric Feature:"),
                        dcc.Dropdown(id='vizdrop',
                                     options=[{'label':i,'value':i} for i
in cat2],
                                     value='Base Pay',placeholder='Select
one...'),html.Br(),
                        dcc.Slider(id='vizslide',
                                       min=10,
                                       max=200,
                                       value=50,
                                       step=1,
                                       marks={f'{i}':i for i in
range(10,200,20)},
                                       tooltip={"placement": "bottom",
"always_visible": False}),
                        html.Div(id='vizout'),
                        ],id='visuals'), html.Br(),

                        html.P('Bar Plot'),
                        html.Div([
                                html.P('Pick the Date Range'),
                               html.Div([
html.P('From:'),dcc.Dropdown(id='date1',
                                        options=[{'label': i, 'value': i}
for i in range(2011,2020,1)],
                                        value=2011),
                                         html.P('To:'),
                                         dcc.Dropdown(id='date2',
                                                      options=[{'label':
i, 'value': i} for i in range(2011, 2020, 1)],
                                                      value=2012)
                                         ],style={'width':
'20%','display': 'inline-block'}),
                                html.P("Select Job Title"),
                                dcc.Dropdown(id='bardrop',
                                  options=[{'label': i, 'value': i} for i
in df_clean['Job Title'].unique()],
                                  placeholder='Select
one...',value='FIREFIGHTER'), html.Br(),
                        ]),
                        html.Div(id='barout'),html.Br(),

                        html.P('Heatmap Plot'),
                        html.Div(children=[
                        dcc.Checklist(id = "heatcheck",
                              options=[{'label': i, 'value': i} for i in
cat2],
                             value=["Base Pay","Benefits"]),
```

```
                            html.Br(),
                            ]),
                            html.Div(id='heatmap'),html.Br(),

                            html.P('Correlation Matrix Plot'),
                            dcc.Checklist(id = "corrcheck",
                                options=[{'label': i, 'value': i} for i in
cat2],
                               value=["Base Pay","Benefits"]),
                            html.Div(id='corrout'),html.Br(),

                            html.P('LINE PLOT'),html.Br(),
                            html.P('Select Feature:'),
                            dcc.Dropdown(id='linedrop',
                                         options=[{'label': i, 'value': i} for
i in cat2],
                                         value='Base Pay', placeholder='Select
one...',multi=False), html.Br(),

                            html.P('Pick Year'),
                                   html.Div([ dcc.Dropdown(id='date3',
                                             options=[{'label': i, 'value': i}
for i in range(2011,2020,1)],
                                             value=2011),
                            html.Div(id='lineout')]),html.Br(),

                            html.P('COUNT PLOT'),html.Br(),
                            html.Div(
                                dcc.Graph(figure=figcount)
                            ),

                            html.P('BOX PLOT'),html.Br(),
                            html.P('Select Feature:'),
                            dcc.Dropdown(id='boxdrop',
                                         options=[{'label': i, 'value': i} for
i in cat2],
                                         value='Base Pay', placeholder='Select
one...',multi=False), html.Br(),
                            html.Div(id='boxxout'),html.Br(),

                            html.P('VIOLIN PLOT'),html.Br(),
                            html.P('Select Feature:'),
                            dcc.Dropdown(id='viodrop',
                                         options=[{'label': i, 'value': i} for
i in cat2],
                                         value='Base Pay', placeholder='Select
one...',multi=False), html.Br(),
                            html.Div(id='vioout'),

                            html.P('REGRESSION PLOT'),html.Br(),
                            html.P('Select Feature to regress on:'),
                            dcc.Dropdown(id='regdrop',
                                         options=[{'label': i, 'value': i} for
i in cat2],
```

```
                                          value='Base Pay', placeholder='Select
one...',multi=False),
                            html.Div(id='regout'),

])

@my_app.callback(
    Output('g1','figure'),
    [Input('filter','value')]
)
def update(f):
    if f== 'IQR':
        fig = make_subplots(rows=1, cols=2)
        fig.add_trace(
            go.Histogram(x=df_no_outlier['Total Pay &
Benefits'],name='Total Pay & Benefits'),
            row=1, col=1)
        fig.add_trace(
            go.Box(y=df_no_outlier['Total Pay & Benefits'],name='Total Pay
& Benefits'),
            row=1, col=2)
        fig.update_layout(
            title_text='Histogram & Box Plot of Total Pay & Benefits',   #
title of plot
        )
        return fig
    else:
        fig = make_subplots(rows=1, cols=2)
        fig.add_trace(
            go.Histogram(x=df_clean['Total Pay & Benefits'],name='Total
Pay & Benefits'),
            row=1, col=1)
        fig.add_trace(
            go.Box(y=df_clean['Total Pay & Benefits'],name='Total Pay &
Benefits'),
            row=1, col=2)
        fig.update_layout(
            title_text='Histogram & Box Plot of Total Pay & Benefits',   #
title of plot
        )
        return fig


@my_app.callback(
    Output('normal-out','children'),
    [Input("normdrop","value"),
     Input('normradio','value'),Input('transform','value')]
)
def update(d,r,t):
    quantil = QuantileTransformer(output_distribution='normal')
    r= df_clean if r=="with-outliers" else df_no_outlier
    quantile_t=quantil.fit_transform(r['Total Pay &
Benefits'].values.reshape(-1,1))
    box,_=stats.boxcox(r['Total Pay & Benefits'].values)
```

```
    sqr= np.sqrt(r['Total Pay & Benefits'].values)
    reciprocal= 1/r['Total Pay & Benefits'].values
    if d== 'shapiro':
        if t== 'quantile':
            return html.Div(str(shapiro_test(quantile_t, 'Total Pay &
Benefits')))
        elif t== 'box':
            return html.Div(str(shapiro_test(box, 'Total Pay &
Benefits')))
        elif t== 'sqrt':
            return html.Div(str(shapiro_test(sqr, 'Total Pay &
Benefits')))
        elif t == 'reciprocal':
            return html.Div(str(shapiro_test(reciprocal, 'Total Pay &
Benefits')))
        else:
            return html.Div(str(shapiro_test(r['Total Pay & Benefits'],
'Total Pay & Benefits')))
    elif d== 'ks':
        if t== 'quantile':
            return html.Div(str(ks_test(quantile_t, 'Total Pay &
Benefits')))
        elif t== 'box':
            return html.Div(str(ks_test(box, 'Total Pay & Benefits')))
        elif t== 'sqrt':
            return html.Div(str(ks_test(sqr, 'Total Pay & Benefits')))
        elif t== 'reciprocal':
            return html.Div(str(ks_test(reciprocal, 'Total Pay &
Benefits')))
        else:
            return html.Div(str(ks_test(r['Total Pay & Benefits'], 'Total
Pay & Benefits')))
    elif d== 'dap':
        if t== 'quantile':
            return html.Div(str(da_k_squared_test(quantile_t, 'Total Pay &
Benefits')))
        elif t== 'box':
            return html.Div(str(da_k_squared_test(box, 'Total Pay &
Benefits')))
        elif t== 'sqrt':
            return html.Div(str(da_k_squared_test(sqr, 'Total Pay &
Benefits')))
        elif t == 'reciprocal':
            return html.Div(str(da_k_squared_test(reciprocal, 'Total Pay &
Benefits')))
        else:
            return html.Div(str(da_k_squared_test(r['Total Pay &
Benefits'], 'Total Pay & Benefits')))
    elif d=='hist':
        if t== 'quantile':
            return
dcc.Graph(figure=px.histogram(x=quantile_t.ravel(),nbins=50,template="plot
ly_dark"))
        elif t== 'box':
```

```
            return
dcc.Graph(figure=px.histogram(x=box,nbins=50,template="plotly_dark"))
        elif t== 'sqrt':
            return
dcc.Graph(figure=px.histogram(x=sqr,nbins=50,template="plotly_dark"))
        elif t == 'reciprocal':
            return
dcc.Graph(figure=px.histogram(x=reciprocal,nbins=50,template="plotly_dark"
))
        else:
            return dcc.Graph(figure=px.histogram(x=r['Total Pay &
Benefits'], nbins=50,template="plotly_dark"))
    else:
        if t== 'quantile':
            return dcc.Graph(figure=qqp(quantile_t.ravel()))
        elif t== 'box':
            return dcc.Graph(figure=qqp(box))
        elif t== 'sqrt':
            return dcc.Graph(figure=qqp(sqr))
        elif t == 'reciprocal':
            return dcc.Graph(figure=qqp(reciprocal))
        else:
            return dcc.Graph(figure=qqp(r['Total Pay & Benefits']))
@my_app.callback(
    Output('overview','children'),
    [Input('overcheck','value'),
     Input('overcheck2','value'),
     Input('overradio','value'),
     Input('overin','value')]
)
def update(a,b,c,d):
    c = df_clean if c == "with-outliers" else df_no_outlier
    fig = px.pie(c[(c['Year']==2011) | (c['Year']==2012)
|(c['Year']==2013) ], values=a, names=b,facet_col='Year',
                 hole=float(d) ,title=f"Pie Chart for {a} with respect to
{b}",

color_discrete_sequence=px.colors.sequential.RdBu,template="plotly_dark")
    fig2 = px.pie(c[(c['Year'] == 2014) | (c['Year'] == 2015) | (c['Year']
== 2016)], values=a, names=b, facet_col='Year',
                 hole=float(d),

color_discrete_sequence=px.colors.sequential.RdBu,template="plotly_dark")
    fig3 = px.pie(c[(c['Year'] == 2017) | (c['Year'] == 2018) | (c['Year']
== 2019)], values=a, names=b,
                  facet_col='Year',
                  hole=float(d),

color_discrete_sequence=px.colors.sequential.RdBu,template="plotly_dark")
    return
html.Div([dcc.Graph(figure=fig),dcc.Graph(figure=fig2),dcc.Graph(figure=fi
g3)])

@my_app.callback(
```

```
    Output('vizout','children'),
    [Input('vizradio','value'),Input('vizdrop','value'),
     Input('vizslide','value')]
)
def update(a,b,c):
    dftitle=a
    a = df_clean if a == "with-outliers" else df_no_outlier
    fig = px.histogram(x=a[b], nbins=int(c),title=f"Histogram Plot for {b}
{[str(dftitle)]}",template="plotly_dark")
    return dcc.Graph(figure=fig)

@my_app.callback(
    Output('barout','children'),
    [Input('vizradio','value'),
     Input('bardrop','value'),
     Input('date1','value'),
     Input('date2','value')]
)
def update(a,b,c,d):
    dftitle = a
    a = df_clean if a == "with-outliers" else df_no_outlier
    df=a[a['Job Title']==b]
    df= df[(df['Year']==int(c)) | (df['Year']==int(d))]
    df=df[cat2].sum()
    fig = px.bar(x=df.index, y=df, text=df,title=f"Bar-Plot for {b}
{[dftitle]}",template="plotly_dark")
    fig.update_traces(texttemplate='%{text:.2s}', textposition='outside')
    fig.update_layout(uniformtext_minsize=8, uniformtext_mode='hide')
    return dcc.Graph(figure=fig)

@my_app.callback(
    Output('heatmap','children'),
    [Input('vizradio','value'),Input('heatcheck','value')]
)
def update(a,b):
    dftitle = a
    a = df_clean if a == "with-outliers" else df_no_outlier
    df=a[b].corr()
    fig = px.imshow(df, text_auto=True,
                    title=f"Heatmap on Numeric Features {[dftitle]}",
                    color_continuous_scale=px.colors.sequential.Cividis_r,
                    template="plotly_dark")
    return dcc.Graph(figure=fig)

@my_app.callback(
    Output('corrout','children'),
    [Input('vizradio','value'),Input('corrcheck','value')]
)
def update(a,b):
    dftitle = a
    a = df_clean if a == "with-outliers" else df_no_outlier
    # fig = px.scatter_matrix(a[b],color=a.Status)
    fig = px.scatter_matrix(a,dimensions=b,
color='Status',symbol='Status',
```

```
                              title=f"Scatter Matrix on Numeric Features Per
Job Status{[dftitle]}",template="plotly_dark")
    fig.update_traces(diagonal_visible=False)

    return dcc.Graph(figure=fig)


@my_app.callback(
    Output('lineout','children'),
    [Input('vizradio','value'),Input('linedrop','value'),
     Input('date3','value')]
)
def update(a,b,year):
    dftitle = a
    a = df_clean if a == "with-outliers" else df_no_outlier
    a = a[a.Year == int(year)]
    fig = px.line(a,y=b,title=f"Line Plot for {b} for Year {year}",
                  template="plotly_dark")
    # fig= go.Figure(data=go.Scatter(x=a.Year, y=a[cat2]))
    return dcc.Graph(figure=fig)


@my_app.callback(
    Output('boxxout','children'),
    [Input('vizradio','value'),Input('boxdrop','value')]
)
def update(a,b):
    dftitle = a
    a = df_clean if a == "with-outliers" else df_no_outlier
    fig = px.box(a, x="Year", y=b, color="Status",template="plotly_dark",
                 title=f"Box Plot for {b} {[dftitle]}")
    return dcc.Graph(figure=fig)

@my_app.callback(
    Output('vioout','children'),
    [Input('vizradio','value'),Input('viodrop','value')]
)
def update(a,b):
    dftitle = a
    a = df_clean if a == "with-outliers" else df_no_outlier
    fig = px.violin(a, x="Year", y=b,
color="Status",template="plotly_dark",
                 title=f"Box Plot for {b} {[dftitle]}")
    return dcc.Graph(figure=fig)

@my_app.callback(
    Output('regout','children'),
    [Input('vizradio','value'),Input('regdrop','value')]
)
def update(a,b):
    dftitle = a
    a = df_clean if a == "with-outliers" else df_no_outlier
    fig = px.scatter(a, y="Total Pay & Benefits", x=b,
                     trendline="ols",template="plotly_dark",
```

```
                      title=f"Regression Plot for Total Pay & Benefits Vs
{b}")
    return dcc.Graph(figure=fig)




# Please Note: host='127.0.0.1' works for me else host='0.0.0.0' Thank
you.
if __name__ == '__main__':
    my_app.run_server(
        port = random.randint(8000,9999), #8080
        host = "127.0.0.1"
    )

#df_clean[df_clean['Job Title']=="Electrical Transit System
Mech"][cat2].sum()
#df_clean['Job Title'].unique()
#int(g[:4])
```