Get dataset

```
# import requried libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# df=pd.read_csv("")
df=pd.read_csv("ds_salaries.csv")
df.head()
```

	Unnamed: 0	work_year	experience_level	employment_type	job_title	salary	salary_cu
0	0	2020	MI	FT	Data Scientist	70000	
1	1	2020	SE	FT	Machine Learning Scientist	260000	
2	2	2020	SE	FT	Big Data Engineer	85000	
3	3	2020	MI	FT	Product Data Analyst	20000	
4	4	2020	SE	FT	Machine Learning Engineer	150000	

Data Preprocessing

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 607 entries, 0 to 606
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Unnamed: 0          607 non-null    int64
```

```
 1   work_year           607 non-null    int64
 2   experience_level    607 non-null    object
 3   employment_type     607 non-null    object
 4   job_title           607 non-null    object
 5   salary              607 non-null    int64
 6   salary_currency     607 non-null    object
 7   salary_in_usd       607 non-null    int64
 8   employee_residence  607 non-null    object
 9   remote_ratio        607 non-null    int64
 10  company_location    607 non-null    object
 11  company_size        607 non-null    object
dtypes: int64(5), object(7)
memory usage: 57.0+ KB
```

No null value here so we are going further.

```
df.drop('Unnamed: 0',axis=1,inplace=True)       #remove unanmed: 0  bcoz  of no use
df.head()
```

	work_year	experience_level	employment_type	job_title	salary	salary_currency	sa
0	2020	MI	FT	Data Scientist	70000	EUR	
1	2020	SE	FT	Machine Learning Scientist	260000	USD	
2	2020	SE	FT	Big Data Engineer	85000	GBP	
3	2020	MI	FT	Product Data Analyst	20000	USD	
4	2020	SE	FT	Machine Learning Engineer	150000	USD	

```
df.shape
```

```
(607, 11)
```

```
#Experience level-effect a lot on job salary so analysis it more
df['experience_level'].unique()
```

```
array(['MI', 'SE', 'EN', 'EX'], dtype=object)
```

```
df['experience_level'].value_counts()
```

```
SE    280
MI    213
EN     88
EX     26
Name: experience_level, dtype: int64
```

```
#replacing abbrevations
df['experience_level'] = df['experience_level'].map({
    'SE': 'Senior',
    'MI': 'Mid',
    'EN': 'Entry',
    'EX': 'Executive'
})
```

```
#replacing abbrevations
df['experience_level'].unique()
```

```
array(['Mid', 'Senior', 'Entry', 'Executive'], dtype=object)
```

```
# employment type
df['employment_type'].value_counts()
```

```
FT    588
PT     10
CT      5
FL      4
Name: employment_type, dtype: int64
```

```
#replacing abbrevations
df['employment_type'] = df['employment_type'].map({
    'FT': 'Full-time',
    'PT': 'Part-time',
    'CT': 'Contract',
    'FL': 'Freelance'
})
```

```
df['job_title'].unique()
```

```
array(['Data Scientist', 'Machine Learning Scientist',
       'Big Data Engineer', 'Product Data Analyst',
       'Machine Learning Engineer', 'Data Analyst', 'Lead Data Scientist',
       'Business Data Analyst', 'Lead Data Engineer', 'Lead Data Analyst',
       'Data Engineer', 'Data Science Consultant', 'BI Data Analyst',
       'Director of Data Science', 'Research Scientist',
       'Machine Learning Manager', 'Data Engineering Manager',
       'Machine Learning Infrastructure Engineer', 'ML Engineer',
```

```
       'AI Scientist', 'Computer Vision Engineer',
       'Principal Data Scientist', 'Data Science Manager', 'Head of Data',
       '3D Computer Vision Researcher', 'Data Analytics Engineer',
       'Applied Data Scientist', 'Marketing Data Analyst',
       'Cloud Data Engineer', 'Financial Data Analyst',
       'Computer Vision Software Engineer',
       'Director of Data Engineering', 'Data Science Engineer',
       'Principal Data Engineer', 'Machine Learning Developer',
       'Applied Machine Learning Scientist', 'Data Analytics Manager',
       'Head of Data Science', 'Data Specialist', 'Data Architect',
       'Finance Data Analyst', 'Principal Data Analyst',
       'Big Data Architect', 'Staff Data Scientist', 'Analytics Engineer',
       'ETL Developer', 'Head of Machine Learning', 'NLP Engineer',
       'Lead Machine Learning Engineer', 'Data Analytics Lead'],
      dtype=object)
```

```
# company size
df['company_size'].value_counts()
```

```
M    326
L    198
S     83
Name: company_size, dtype: int64
```

```
#replacing abbrevations
df['company_size'] = df['company_size'].map({
    'S': 'Small',
    'M': 'Medium',
    'L': 'Large'
})
```

```
# drop salary and salary_currency features (salary_in_usd is enough to keep on)
df.drop(['salary', 'salary_currency'], axis=1, inplace=True)

# rename salary_in_usd to salary
df.rename(columns={'salary_in_usd': 'salary'}, inplace=True)
```

```
#work year
df['work_year'].value_counts()
```

```
2022    318
2021    217
2020     72
Name: work_year, dtype: int64
```

```
#remote ratio
df['remote_ratio'].value_counts()
```

```
100    381
0      127
```

```
50      99
Name: remote_ratio, dtype: int64
```

```
# renmame remote_ratio to job_type
df.rename(columns={'remote_ratio': 'job_type'}, inplace=True)

# change 100 to remote, 0 to onsite, 50 to hybrid
df['job_type'] = df['job_type'].map({
    100: 'remote',
    0: 'onsite',
    50: 'hybrid',
})
```

```
df['job_type'].value_counts()
```

```
remote    381
onsite    127
hybrid     99
Name: job_type, dtype: int64
```

```
df.columns
```

```
Index(['work_year', 'experience_level', 'employment_type', 'job_title',
       'salary', 'employee_residence', 'job_type', 'company_location',
       'company_size'],
      dtype='object')
```

```
df.head()
```

Now we get good data-set but we can't understand 'employee_residence' and 'company_location' so we change the short form of country name to full form by ISO Country code

```
# change country names from ISO2 to original names
# There are two features containing country names, "company_location" and "employee_residence
# country converter-https://vincentarelbundock.github.io/countrycode/
!pip install country_converter
import country_converter
cc = country_converter.CountryConverter()
df['company_location'] = cc.convert(df['company_location'], to='name_short')
df['employee_residence'] = cc.convert(df['employee_residence'], to='name_short')
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Requirement already satisfied: country_converter in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: pandas>=1.0 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (
```

```
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from
```

```
df.head()
```

Extract Data Analysis

```
df.describe()
```

	work_year	salary
count	607.000000	607.000000
mean	2021.405272	112297.869852
std	0.692133	70957.259411
min	2020.000000	2859.000000
25%	2021.000000	62726.000000
50%	2022.000000	101570.000000
75%	2022.000000	150000.000000
max	2022.000000	600000.000000

mean salary is $112297

max salary is $600000

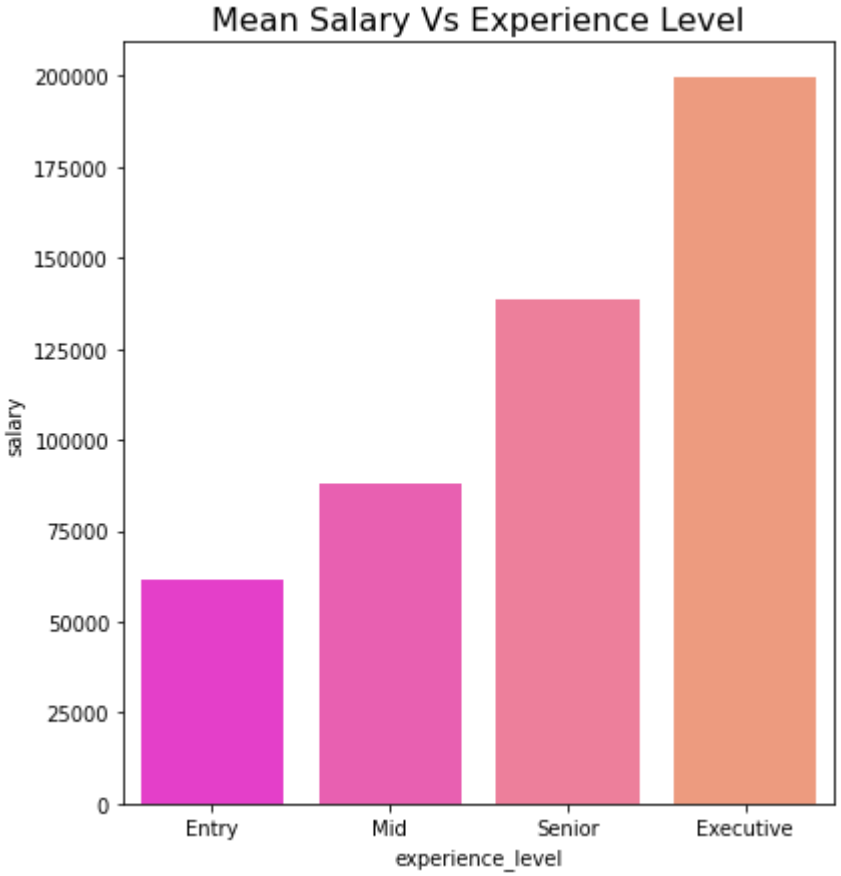
```
#now we see graph beetween experince v/s salary
exp_salary = df.groupby('experience_level')['salary'].mean().sort_values()
exp_salary
```

```
experience_level
Entry         61643.318182
Mid           87996.056338
Senior       138617.292857
Executive    199392.038462
Name: salary, dtype: float64
```

```
plt.figure(figsize=(14, 7))
sns.set_palette('spring')
```

```
plt.subplot(1, 2, 1)
ax = sns.barplot(x=exp_salary.index, y=exp_salary)
ax.set_title('Mean Salary Vs Experience Level', fontdict={'fontsize': 16})
```

```
Text(0.5, 1.0, 'Mean Salary Vs Experience Level')
```



Salary v/s Employement type

```
salary_emp_type = df.groupby('employment_type')['salary'].mean().sort_values()
salary_emp_type
```

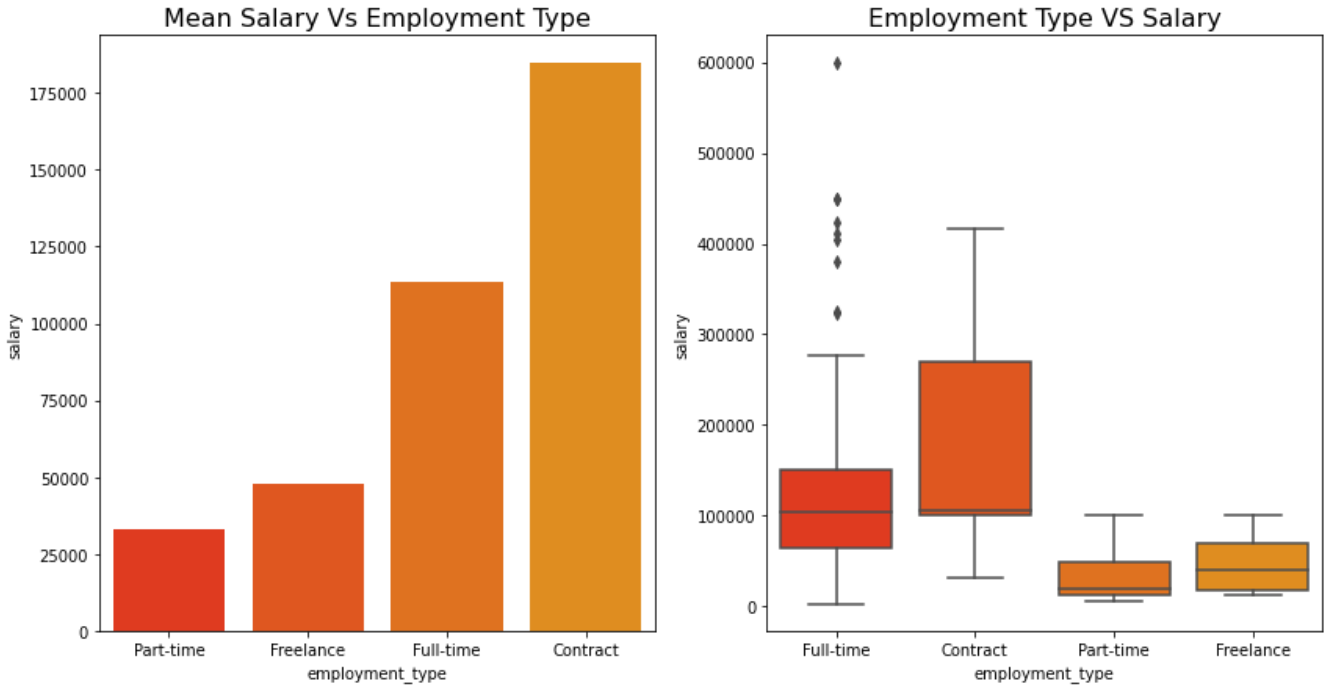
```
employment_type
Part-time     33070.500000
Freelance     48000.000000
Full-time    113468.073129
Contract     184575.000000
Name: salary, dtype: float64
```

FOUND-contract base salary is highest.

```
plt.figure(figsize=(14, 7))
sns.set_palette('autumn')
```

```
plt.subplot(1, 2, 1)
ax = sns.barplot(x=salary_emp_type.index, y=salary_emp_type)
ax.set_title('Mean Salary Vs Employment Type', fontdict={'fontsize': 16})

plt.subplot(1, 2, 2)
ax = sns.boxplot(data=df, x='employment_type', y='salary')
ax.set_title('Employment Type VS Salary', fontdict={'fontsize': 16})
```

```
Text(0.5, 1.0, 'Employment Type VS Salary')
```



FOUND-contract base salary is highest and part time salary is lowest.

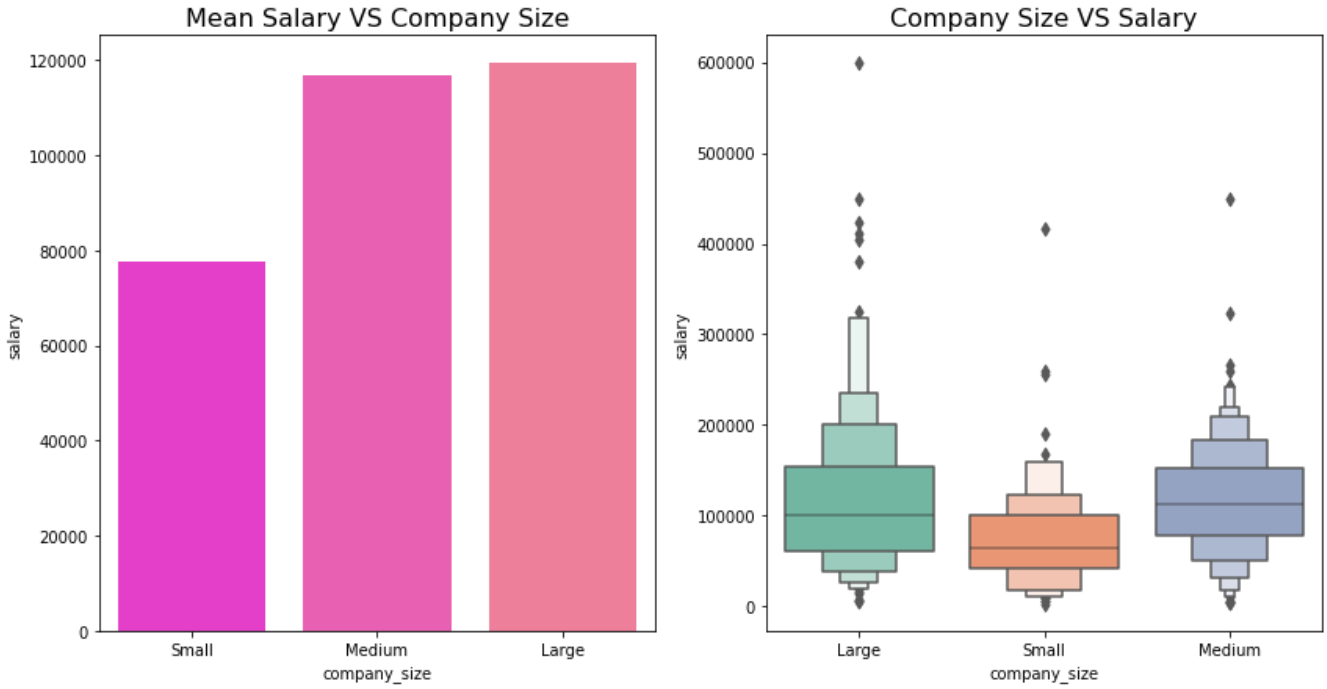
Salary VS Company Size

```
# mean salary of employees from different company sizes
salary_cmp_size = df.groupby('company_size')['salary'].mean().sort_values()
salary_cmp_size
```

```
company_size
Small      77632.674699
Medium    116905.466258
Large     119242.994949
Name: salary, dtype: float64
```

```
plt.figure(figsize=(14, 7))
sns.set_palette('spring')

plt.subplot(1, 2, 1)
ax = sns.barplot(x=salary_cmp_size.index, y=salary_cmp_size)
ax.set_title('Mean Salary VS Company Size', fontdict={'fontsize': 16})

plt.subplot(1, 2, 2)
sns.set_palette('Set2')
ax = sns.boxenplot(data=df, x='company_size', y='salary')
ax.set_title('Company Size VS Salary', fontdict={'fontsize': 16})
```

```
Text(0.5, 1.0, 'Company Size VS Salary')
```



Company Size VS Salary: Large companies have highest mean salary and those who work at small ones have the lowest mean salary.

Salary VS Job type (remote, hybrid, onsite)

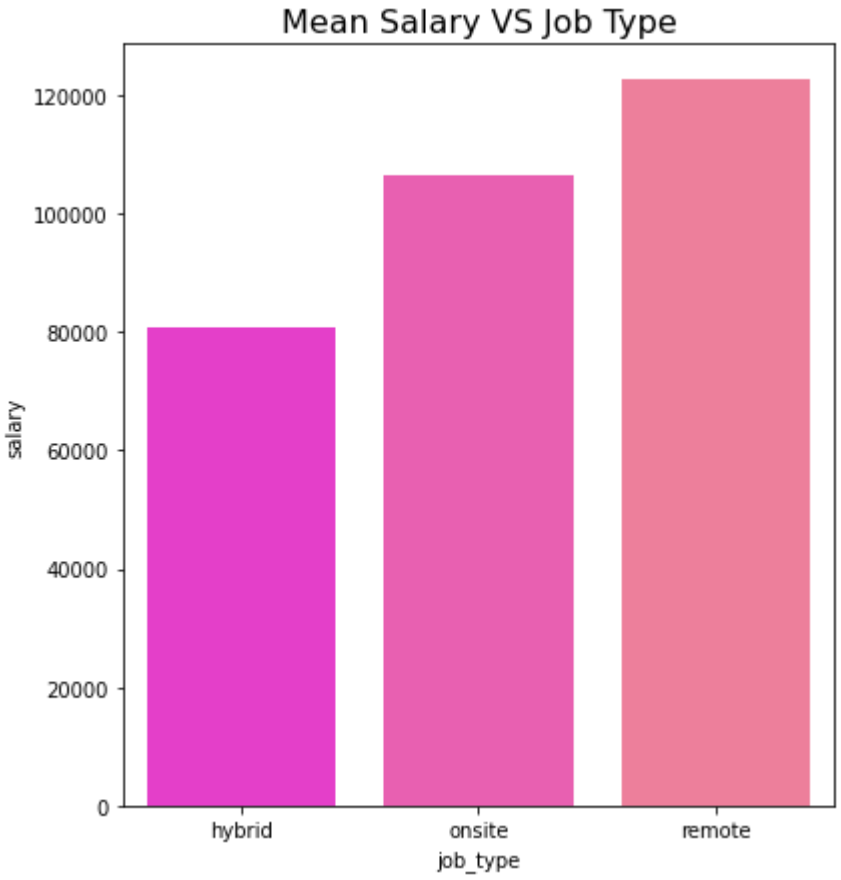
```
# mean salary of employees with different job types
salary_jtype = df.groupby('job_type')['salary'].mean().sort_values()
salary_jtype
```

```
job_type
```

```
hybrid     80823.030303
onsite    106354.622047
remote    122457.454068
Name: salary, dtype: float64
```

```
plt.figure(figsize=(14, 7))
sns.set_palette('spring')

plt.subplot(1, 2, 1)
ax = sns.barplot(x=salary_jtype.index, y=salary_jtype)
ax.set_title('Mean Salary VS Job Type', fontdict={'fontsize': 16})
```

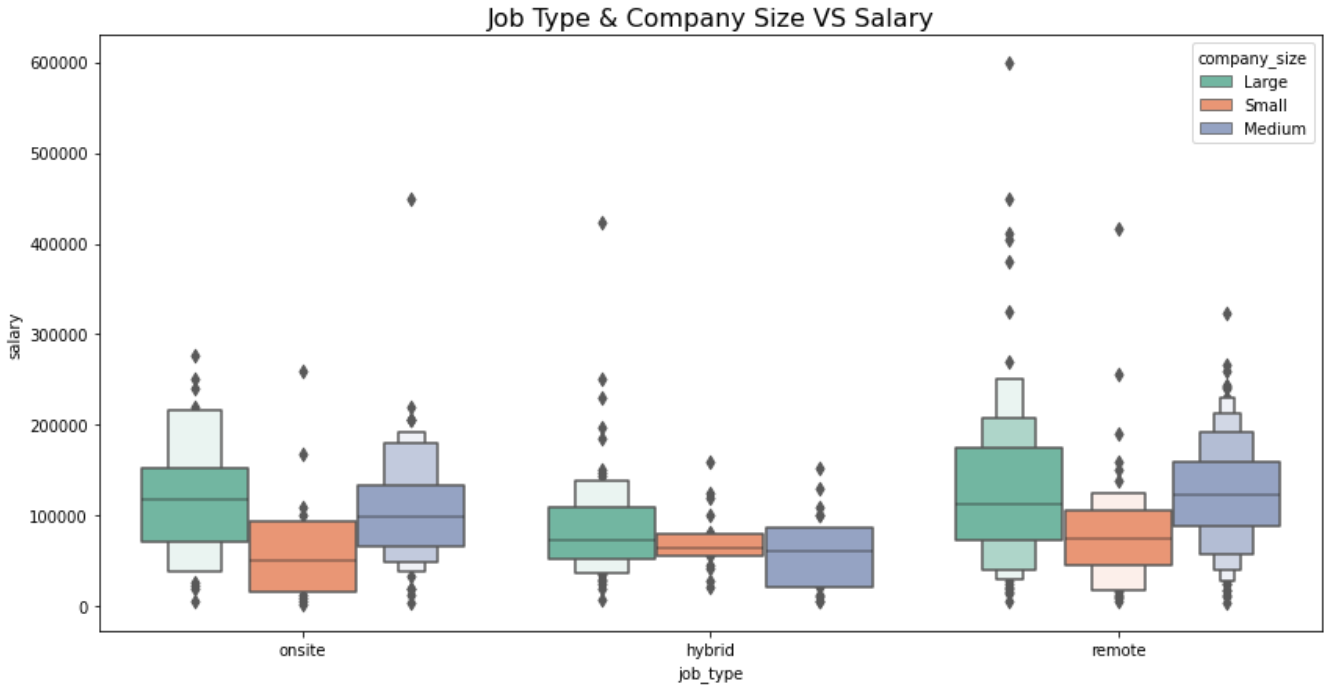
```
Text(0.5, 1.0, 'Mean Salary VS Job Type')
```



Job Type (remote, on-site or hybrid) VS Salary: remotely have a higher mean salary than those who work on-site, and hybrid workers have a lower mean salary.

```
# job type and company size VS salary
plt.figure(figsize=(14, 7))
sns.set_palette('Set2')
ax = sns.boxenplot(data=df, x='job_type', y='salary', hue='company_size')
ax.set_title('Job Type & Company Size VS Salary', fontdict={'fontsize': 16})
```

```
Text(0.5, 1.0, 'Job Type & Company Size VS Salary')
```

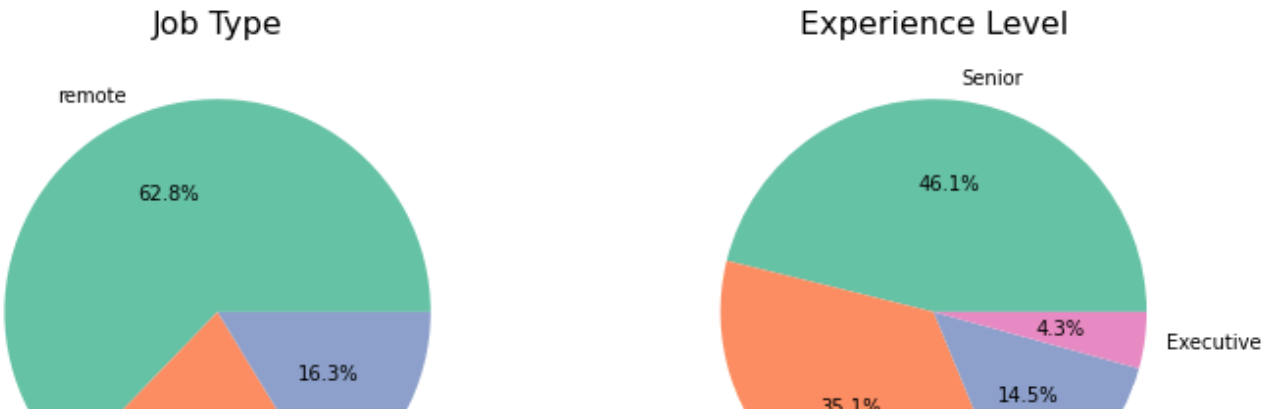


so, prefer to work remotely at a large company to get higher salary.

Job Types and Experience Level distributions (Pie)

```
plt.figure(figsize=(12, 5))
sns.set_palette('Set2')

# job types
plt.subplot(1,2,1)
ax = df['job_type'].value_counts().plot(kind='pie', autopct='%1.1f%%')
ax.set_title('Job Type', fontdict={'fontsize': 16})
ax.set_ylabel('')

# experience levels
plt.subplot(1,2,2)
ax = df['experience_level'].value_counts().plot(kind='pie', autopct='%1.1f%%')
ax.set_title('Experience Level', fontdict={'fontsize': 16})
ax.set_ylabel('')
```

```
Text(0, 0.5, '')
```



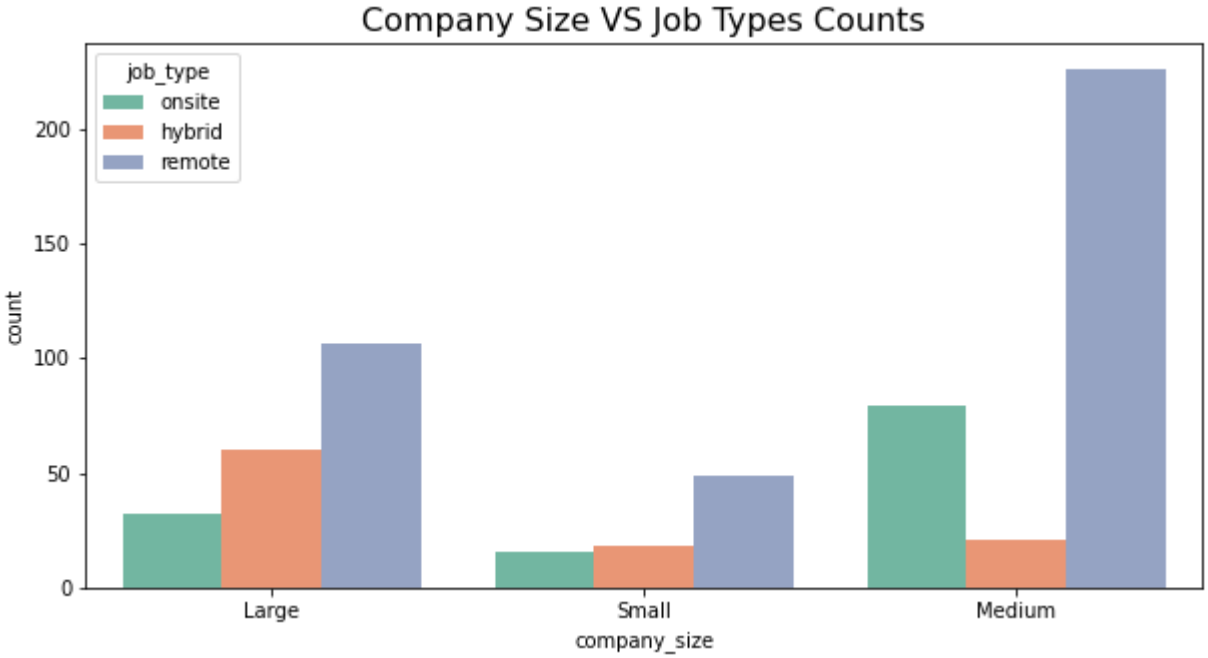
-> Remote jobs have the highest persent of openings.

-> Share of Job openings for employees with an experience level of Senior is the the highest here.

Company Size VS Job Types Counts

```
plt.figure(figsize=(10, 5))
sns.set_palette('Set2')
ax = sns.countplot(data=df, x='company_size', hue='job_type')
ax.set_title('Company Size VS Job Types Counts', fontdict={'fontsize': 16})
```

```
Text(0.5, 1.0, 'Company Size VS Job Types Counts')
```



-> In all companies, the number of remote workers is higher than that of hybrid and on-site.

-> The number of hybrid workers in small and large companies is higher than that of on-site, whereas in medium-sized companies, more people work on-site than hybrid.

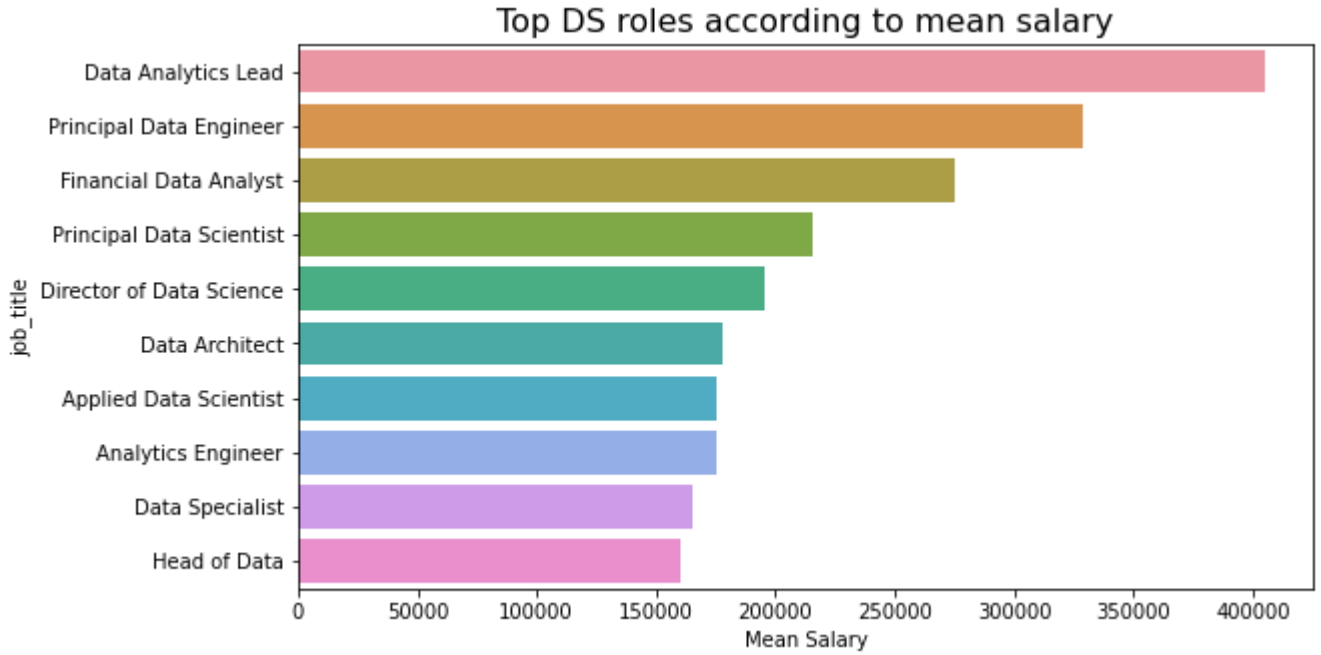
Top 10 Data Science Roles

```
# top 10 data science roles according to mean salary
top_ds_roles = df.groupby('job_title')['salary'].mean().sort_values(ascending=False)
top_ds_roles=top_ds_roles.head(10)
top_ds_roles
```

```
job_title
Data Analytics Lead         405000.000000
Principal Data Engineer     328333.333333
Financial Data Analyst      275000.000000
Principal Data Scientist    215242.428571
Director of Data Science    195074.000000
Data Architect              177873.909091
Applied Data Scientist      175655.000000
Analytics Engineer          175000.000000
Data Specialist             165000.000000
Head of Data                160162.600000
Name: salary, dtype: float64
```

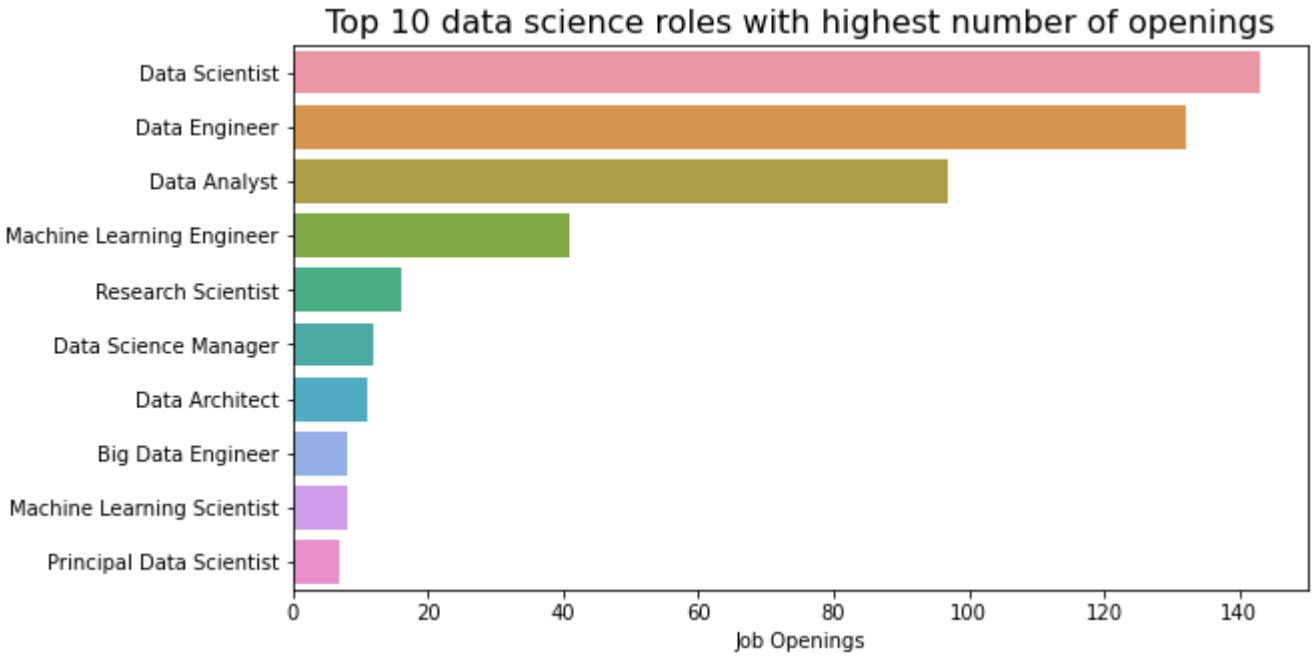
```
# top 10 data science roles according to mean salary and counts
plt.figure(figsize=(20, 5))
plt.subplot(1, 2, 1)
top_ds_roles = top_ds_roles
ax = sns.barplot(y=top_ds_roles.index, x=top_ds_roles)
ax.set_xlabel('Mean Salary')
ax.set_title('Top DS roles according to mean salary', fontdict={'fontsize': 16})
```

```
Text(0.5, 1.0, 'Top DS roles according to mean salary')
```



```
# top 10 data science roles with highest number of openings
plt.figure(figsize=(20, 5))
plt.subplot(1, 2, 2)
top_dr = df['job_title'].value_counts()[:10]

ax = sns.barplot(x=top_dr, y=top_dr.index)
ax.set_xlabel('Job Openings')
ax.set_title('Top 10 data science roles with highest number of openings', fontdict={'fontsize
```

```
Text(0.5, 1.0, 'Top 10 data science roles with highest number of openings')
```



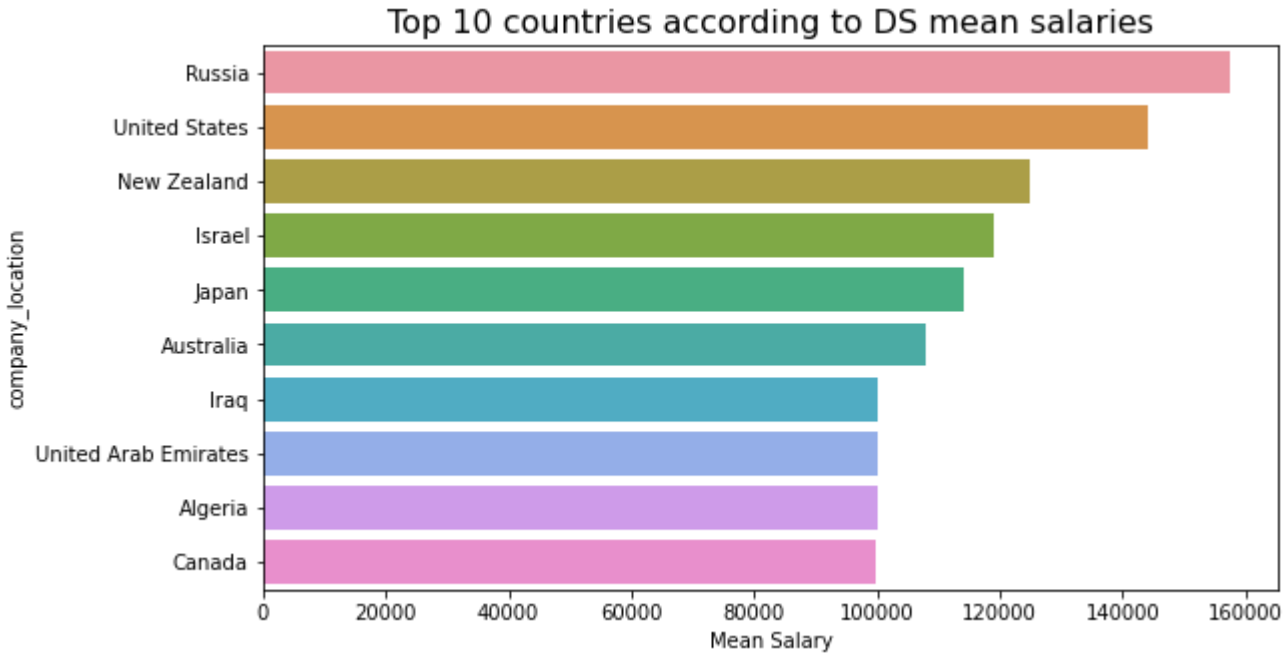
Top 10 campany-locations

```
# top 10 company-locations according to mean salary
top_cmp_locations = df.groupby('company_location')['salary'].mean().sort_values(ascending=Fal
top_cmp_locations
```

```
company_location
Russia                  157500.000000
United States           144055.261972
New Zealand             125000.000000
Israel                  119059.000000
Japan                   114127.333333
Australia               108042.666667
Iraq                    100000.000000
United Arab Emirates    100000.000000
Algeria                 100000.000000
Canada                   99823.733333
Name: salary, dtype: float64
```

```
# top 10 company-locations according to mean salary
plt.figure(figsize=(20, 5))
plt.subplot(1, 2, 1)
ax = sns.barplot(y=top_cmp_locations.index, x=top_cmp_locations)
ax.set_xlabel('Mean Salary')
ax.set_title('Top 10 countries according to DS mean salaries', fontdict={'fontsize': 16})
```

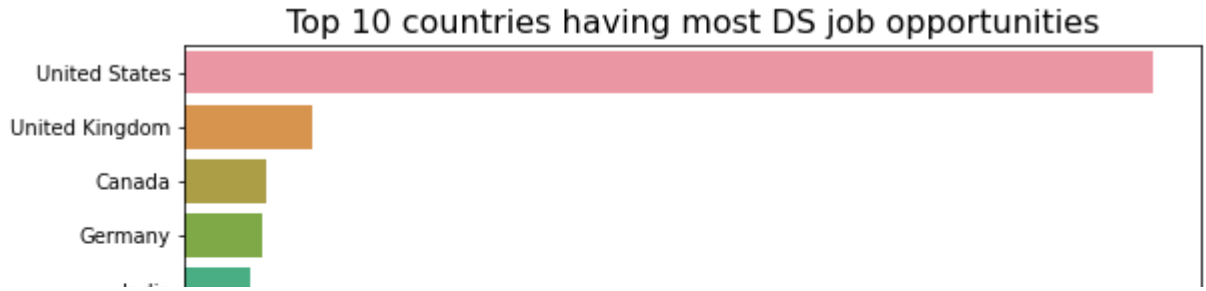
```
Text(0.5, 1.0, 'Top 10 countries according to DS mean salaries')
```



Russia, the United States and New Zealand are the highest paying countries for data science roles according to this dataset.

```
# top 10 company-locations having most job opportunities
plt.figure(figsize=(20, 5))
top_cl = df['company_location'].value_counts()[:10]
plt.subplot(1, 2, 2)
ax = sns.barplot(x=top_cl, y=top_cl.index)
ax.set_xlabel('Number of Job Opportunities')
ax.set_title('Top 10 countries having most DS job opportunities', fontdict={'fontsize': 16})
```

```
Text(0.5, 1.0, 'Top 10 countries having most DS job opportunities')
```



The US, The UK and Canada are the top three countries offering highest number of Data Science job.

Linear Regression Algorithm


```
df.columns
```

```
Index(['work_year', 'experience_level', 'employment_type', 'job_title',
       'salary', 'employee_residence', 'job_type', 'company_location',
       'company_size'],
      dtype='object')
```

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df.shape
```

```
(607, 9)
```

```
X = df[['work_year', 'experience_level', 'employment_type', 'job_title', 'employee_residence'
```

```
X = pd.get_dummies(data=X, drop_first=True)
X.head()
```

	work_year	experience_level_Executive	experience_level_Mid	experience_level_Senio
0	2020	0	1	
1	2020	0	0	
2	2020	0	0	
3	2020	0	1	

```
Y = df['salary']
Y.head()
```

```
0     79833
1    260000
2    109024
3     20000
4    150000
Name: salary, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.7, test_size=0.3, rand
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(424, 165)
(183, 165)
(424,)
(183,)
```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)
```

```
LinearRegression()
```

```
# print the intercept
print(model.intercept_)
```

```
22699571.185054045
```

```
coeff_parameter = pd.DataFrame(model.coef_,X.columns,columns=['Coefficient'])
coeff_parameter
```

	Coefficient
work_year	-1.133704e+04
experience_level_Executive	1.109111e+05
experience_level_Mid	3.542231e+04
experience_level_Senior	5.231098e+04
employment_type_Freelance	-2.573769e+05
...	...
company_location_United Kingdom	1.780406e+17
company_location_United States	1.780406e+17
company_location_Vietnam	0.000000e+00
company_size_Medium	9.986119e+02

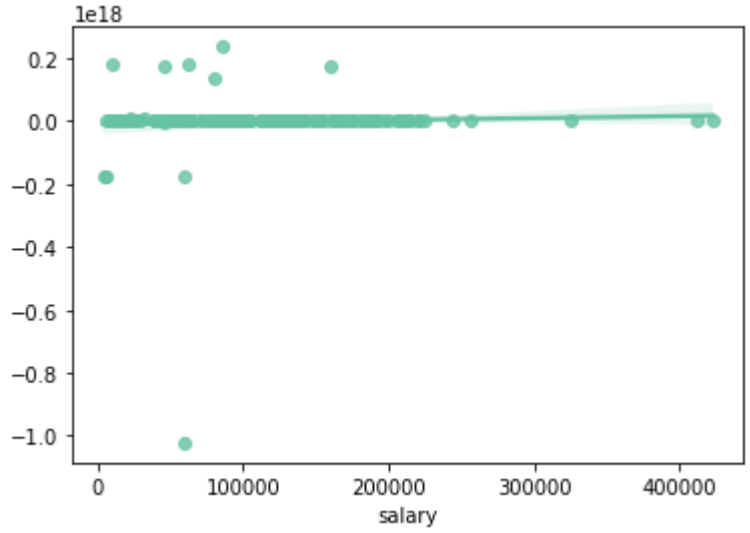
```
predictions = model.predict(X_test)
predictions
```

```
array([ 4.08191851e+04,  1.78040571e+17,  1.05654286e+05,  1.16051185e+05,
        1.16051185e+05,  1.12243185e+05,  5.81900560e+04,  8.82431851e+04,
        6.43123185e+05,  1.50271851e+04,  1.16051185e+05,  1.47827185e+05,
        1.45587185e+05,  5.41767421e+04,  1.46675185e+05,  1.16051185e+05,
        1.35664391e+03,  1.35347185e+05,  1.88659185e+05,  1.47827185e+05,
        9.91551851e+04,  1.99477303e+04,  1.52243185e+05,  1.37107185e+05,
        5.41767421e+04,  9.91551851e+04,  1.46803185e+05, -8.82368149e+04,
       -1.78040571e+17,  1.45587185e+05,  1.47827185e+05, -1.96608149e+04,
        2.01632964e+05,  8.28074714e+04,  9.60191851e+04,  1.16051185e+05,
        1.47827185e+05,  3.14643185e+05,  1.45587185e+05,  1.61107185e+05,
        3.27485975e+05,  1.47827185e+05,  1.45587185e+05,  6.32815637e+04,
        5.41767421e+04,  1.59155185e+05,  1.49011185e+05,  1.67477303e+04,
        9.25311851e+04,  1.20467185e+05,  1.02032930e+05, -1.08189474e+06,
        9.49052555e+04,  1.47827185e+05,  1.36902597e+17,  9.10954714e+04,
        3.47621252e+05,  1.50003185e+05,  5.15434714e+04,  1.18296476e+16,
        1.05843185e+05,  1.44595185e+05,  1.81392423e+17,  3.57990149e+05,
        1.47827185e+05,  1.40243185e+05,  2.30387185e+05,  1.47827185e+05,
        8.60031851e+04,  2.17114469e+05,  1.09491185e+05,  1.33107185e+05,
        8.46911851e+04, -1.78040571e+17,  1.45587185e+05,  1.82694256e+05,
        2.00781947e+05, -1.52448149e+04,  4.57791851e+04,  2.22035185e+05,
        2.12569994e+14,  9.36392828e+04,  1.05651185e+05,  4.52034714e+04,
        1.04595185e+05,  2.38911851e+04, -1.78040571e+17,  9.43172462e+04,
        1.45587185e+05,  2.07731185e+05,  2.32639033e+05,  2.07731185e+05,
        5.74754570e+04,  1.74643185e+05,  4.84351851e+04,  1.23123185e+05,
        4.63865926e+04,  1.47827185e+05,  1.47827185e+05,  1.16051185e+05,
        1.45587185e+05,  1.47827185e+05,  1.47827185e+05, -1.55192704e+07,
        9.91551851e+04,  1.10256542e+05,  1.69651185e+05,  1.35347185e+05,
        1.67667185e+05,  1.83692868e+05,  1.47827185e+05,  3.10231471e+05,
        9.39066435e+04,  2.08691185e+05,  1.67667185e+05,  2.44851185e+05,
        1.47827185e+05,  1.20467185e+05,  8.82431851e+04,  1.68449136e+05,
        1.45587185e+05,  3.30111851e+04,  1.52243185e+05,  7.04234714e+04,
        1.65235185e+05,  2.40345130e+05,  1.20819185e+05,  1.15191471e+05,
        8.60031851e+04,  8.28031851e+04,  1.47827185e+05,  7.96563449e+04,
```

```
       1.43410867e+05,  1.50003185e+05,  9.94111851e+04,  2.82359471e+05,
       1.60339185e+05, -1.53861778e+07,  7.48031851e+04,  1.68663471e+05,
       1.85143987e+12,  5.81209062e+04,  1.67283185e+05,  1.78040568e+17,
      -1.27455459e+05,  5.20831851e+04,  1.50003185e+05,  7.02718505e+03,
       1.81392423e+17,  1.16051185e+05,  1.69459185e+05, -2.89975002e+15,
       1.41267185e+05,  2.55327033e+05,  1.05139185e+05,  1.16051185e+05,
       4.52034714e+04,  6.66669667e+04,  2.37346961e+17,  1.41267185e+05,
       1.69651185e+05,  7.84191851e+04,  1.13907185e+05,  2.24115541e+04,
       8.66878304e+15,  1.05843185e+05,  8.18089005e+04,  1.04595185e+05,
       2.14062916e+04,  1.16051185e+05,  1.50003185e+05,  1.16051185e+05,
       7.58634714e+04,  1.67667185e+05,  4.07724794e+04, -1.02211622e+18,
       7.68617408e+04,  2.18067185e+05,  8.85653684e+04,  1.45587185e+05,
       3.74574717e+04,  8.60031851e+04,  2.83071851e+04])
```

```
sns.regplot(y_test,predictions)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1e63d5b50>
```




```
import statsmodels.api as sm
X_train_Sm= sm.add_constant(X_train)
X_train_Sm= sm.add_constant(X_train)
ls=sm.OLS(y_train,X_train_Sm).fit()
print(ls.summary())
```

```
company_location_Belgium                -4.691e+04   3.26e+04     -1.440
company_location_Brazil                 -9.665e+04   7.16e+04     -1.350
company_location_Canada                 -1.018e+05   7.79e+04     -1.307
company_location_Chile                  -6.099e+04   3.62e+04     -1.683
company_location_China                  -8.305e+04   4.37e+04     -1.902
company_location_Colombia                        0          0        nan
company_location_Croatia                         0          0        nan
company_location_Czech Republic         -6.546e+04   3.69e+04     -1.772
company_location_Denmark                -7263.5993   4.94e+04     -0.147
company_location_Estonia                -6.633e+04   4.16e+04     -1.595
company_location_France                 -5.711e+04   5.14e+04     -1.112
company_location_Germany                -4.151e+04   5.87e+04     -0.707
company_location_Greece                 -3296.3818   6.19e+04     -0.053
company_location_Honduras                        0          0        nan
company_location_Hungary                -8.046e+04   7.65e+04     -1.052
```

```
company_location_Hungary                          -8.040e+04   7.05e+04     -1.052
company_location_India                            -3.614e+04   4.87e+04     -0.742
company_location_Iran                             -7.807e+04   3.63e+04     -2.151
company_location_Iraq                              1.805e+04   3.98e+04      0.453
company_location_Ireland                          -4.906e+04   3.69e+04     -1.328
company_location_Israel                            1.214e+04   4.43e+04      0.274
company_location_Italy                            -6.153e+04   8.88e+04     -0.693
company_location_Japan                             3985.4917   2.97e+04      0.134
company_location_Kenya                                     0          0        nan
company_location_Luxembourg                       -1.589e+04   3.89e+04     -0.408
company_location_Malaysia                         -4.335e+04   3.69e+04     -1.174
company_location_Malta                            -5.393e+04   3.62e+04     -1.491
company_location_Mexico                           -6.967e+04   3.31e+04     -2.107
company_location_Moldova                          -6.388e+04   4.05e+04     -1.578
company_location_Netherlands                      -5.975e+04   2.88e+04     -2.078
company_location_New Zealand                      -2.605e+04   3.97e+04     -0.657
company_location_Nigeria                          -4.816e+04   3.25e+04     -1.481
company_location_Pakistan                         -8.243e+04   1.26e+05     -0.656
company_location_Poland                           -6.363e+04   8.32e+04     -0.765
company_location_Portugal                         -2.707e+04   7.49e+04     -0.361
company_location_Romania                                   0          0        nan
company_location_Russia                           -5.346e+04   6.41e+04     -0.834
company_location_Singapore                        -6.702e+04    5.3e+04     -1.263
company_location_Slovenia                                  0          0        nan
company_location_Spain                            -1.295e+05   5.44e+04     -2.380
company_location_Switzerland                               0          0        nan
company_location_Turkey                           -6.953e+04   3.58e+04     -1.943
company_location_Ukraine                                   0          0        nan
company_location_United Arab Emirates             -2.794e+04   4.37e+04     -0.639
company_location_United Kingdom                   -5329.0799   8.27e+04     -0.064
company_location_United States                    -4.263e+04   3.97e+04     -1.074
company_location_Vietnam                                   0          0        nan
company_size_Medium                                -224.6489   7243.636     -0.031
company_size_Small                                -2.158e+04   1.01e+04     -2.137
==============================================================================
Omnibus:                      133.494   Durbin-Watson:                   2.026
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              952.240
Skew:                           1.148   Prob(JB):                    1.67e-207
Kurtosis:                       9.974   Cond. No.                     4.34e+19
==============================================================================

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

```
from sklearn.metrics import mean_absolute_error


score = mean_absolute_error(y_test, predictions)
print('MAE:', score)
```

```
MAE: 1.4606372005938126e+16
```

classification

```
#Import required libraries
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


#To store results of models, we create two dictionaries
result_dict_train = {}
result_dict_test = {}
```

Applying Logistic Regression classifier

```
#Logistic Regression classifier
reg = LogisticRegression(random_state = 42)
accuracies = cross_val_score(reg, X_train, y_train, cv=5)
reg.fit(X_train,y_train)
y_pred = reg.predict(X_test)

#Obtain accuracy
print("Train Score:",np.mean(accuracies))
print("Test Score:",reg.score(X_test,y_test))
```

```
Train Score: 0.04243697478991597
Test Score: 0.02185792349726776
```

```
#Store results in the dictionaries
result_dict_train["Logistic Train Score"] = np.mean(accuracies)
result_dict_test["Logistic Test Score"] = reg.score(X_test,y_test)
```

Applying K-Nearest Neighbour classifier

```
#K-Nearest Neighbour classifier
knn = KNeighborsClassifier()
accuracies = cross_val_score(knn, X_train, y_train, cv=5)
knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)
#Obtain accuracy
print("Train Score:",np.mean(accuracies))
print("Test Score:",knn.score(X_test,y_test))
```

```
Train Score: 0.037703081232492996
```

```
Test Score: 0.01639344262295082
```

```
#Store results in the dictionaries
result_dict_train["KNN Train Score"] = np.mean(accuracies)
result_dict_test["KNN Test Score"] = knn.score(X_test,y_test)
```

Applying SVC classifier

```
#SVC
svc = SVC(random_state = 42)
accuracies = cross_val_score(svc, X_train, y_train, cv=5)
svc.fit(X_train,y_train)
y_pred = svc.predict(X_test)

#Obtain accuracy
print("Train Score:",np.mean(accuracies))
print("Test Score:",svc.score(X_test,y_test))
```

```
Train Score: 0.021232492997198878
Test Score: 0.01092896174863388
```

```
#Store results in the dictionaries
result_dict_train["SVM Train Score"] = np.mean(accuracies)
result_dict_test["SVM Test Score"] = svc.score(X_test,y_test)
```

Applying Decision Tree Classifier

```
dtc = DecisionTreeClassifier(random_state = 42)
accuracies = cross_val_score(dtc, X_train, y_train, cv=5)
dtc.fit(X_train,y_train)
y_pred = dtc.predict(X_test)

#Obtain accuracy
print("Train Score:",np.mean(accuracies))
print("Test Score:",dtc.score(X_test,y_test))
```

```
Train Score: 0.042464985994397755
Test Score: 0.03278688524590164
```

```
#Store results in the dictionaries
result_dict_train["Decision Tree Train Score"] = np.mean(accuracies)
result_dict_test["Decision Tree Test Score"] = dtc.score(X_test,y_test)
```

Applying Random Forest Classifier

```
rfc = RandomForestClassifier(random_state = 42)
accuracies = cross_val_score(rfc, X_train, y_train, cv=5)
rfc.fit(X_train,y_train)
y_pred = rfc.predict(X_test)
#Obtain accuracy
print("Train Score:",np.mean(accuracies))
print("Test Score:",rfc.score(X_test,y_test))
```

```
Train Score: 0.05190476190476191
Test Score: 0.03278688524590164
```

```
#Store results in the dictionaries
result_dict_train["Random Forest Train Score"] = np.mean(accuracies)
result_dict_test["Random Forest Test Score"] = rfc.score(X_test,y_test)
```

Checking Train Score of all classifier

```
df_result_train = pd.DataFrame.from_dict(result_dict_train,orient = "index", columns=["Score'
df_result_train
```

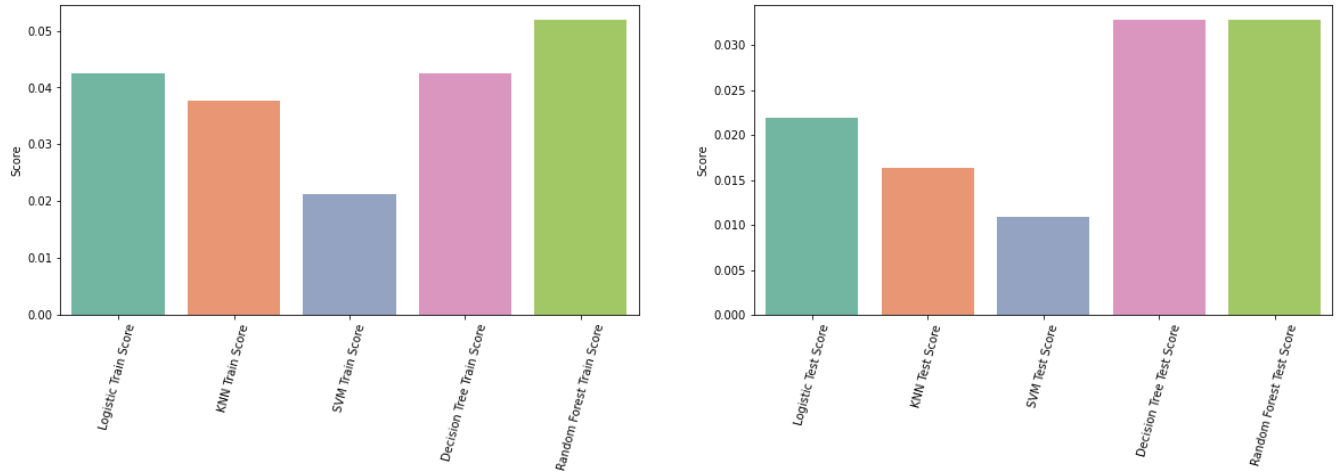
	Score
Logistic Train Score	0.042437
KNN Train Score	0.037703
SVM Train Score	0.021232
Decision Tree Train Score	0.042465
Random Forest Train Score	0.051905

Checking Test Score of all classifier

```
df_result_test = pd.DataFrame.from_dict(result_dict_test,orient = "index",columns=["Score"])
df_result_test
```

Score

```
#barplot visuallization of all classifier
import seaborn as sns
fig,ax = plt.subplots(1,2,figsize=(20,5))
sns.barplot(x = df_result_train.index,y = df_result_train.Score,ax = ax[0])
sns.barplot(x = df_result_test.index,y = df_result_test.Score,ax = ax[1])
ax[0].set_xticklabels(df_result_train.index,rotation = 75)
ax[1].set_xticklabels(df_result_test.index,rotation = 75)
plt.show()
```



Hence, we found Random Forest Regressor have high Score among all classifier.

Clustering

```
# importing necessary libraries
import pandas as pd
import numpy as np
!pip install kmodes
from kmodes.kmodes import KModes
import matplotlib.pyplot as plt
%matplotlib inline
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
```

```
Requirement already satisfied: kmodes in /usr/local/lib/python3.7/dist-packages (0.12.2
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scipy>=0.13.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: scikit-learn>=0.22.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-pa
```

```
df.columns
```

```
Index(['work_year', 'experience_level', 'employment_type', 'job_title',
       'salary', 'employee_residence', 'job_type', 'company_location',
       'company_size'],
      dtype='object')
```

```
# Elbow curve to find optimal K
cost = []
K = range(1,5)
for num_clusters in list(K):
    kmode = KModes(n_clusters=num_clusters, init = "random", n_init = 5, verbose=1)
    kmode.fit_predict(df[['experience_level', 'salary','job_type']])
    cost.append(kmode.cost_)

plt.plot(K, cost, 'bx-')
plt.xlabel('No. of clusters')
plt.ylabel('Cost')
plt.title('Elbow Method For Optimal k')
plt.show()
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 1145.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 0, cost: 1145.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 0, cost: 1145.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 0, cost: 1145.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 1145.0
Best run was number 1
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 7, cost: 928.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 3, cost: 975.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 8, cost: 928.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 99, cost: 972.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 1144.0
Best run was number 1
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 9, cost: 897.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 65, cost: 872.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 97, cost: 923.0
Init: initializing centroids
Init: initializing clusters
```

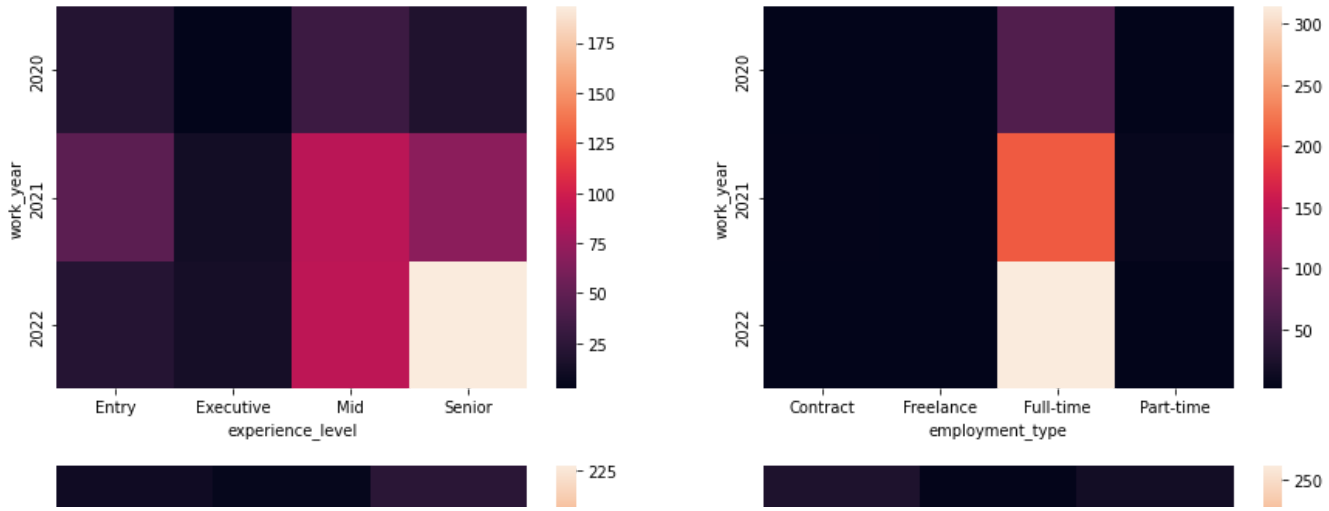
```
Starting iterations...
Run 4, iteration: 1/100, moves: 7, cost: 895.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 4, cost: 1069.0
Best run was number 2
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 86, cost: 839.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 33, cost: 800.0
Run 2, iteration: 2/100, moves: 2, cost: 800.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 5, cost: 839.0
Init: initializing centroids
Init: initializing clusters
```

We can see a bend at K=2 in the above graph indicating 3is the optimal number of clusters.

```
Init: initializing centroids
```

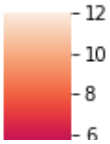
```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.heatmap(pd.crosstab(df['work_year'],df['experience_level']))
plt.subplot(2,2,2)
sns.heatmap(pd.crosstab(df['work_year'],df['employment_type']))
plt.subplot(2,2,3)
sns.heatmap(pd.crosstab(df['work_year'],df['job_type']))
plt.subplot(2,2,4)
sns.heatmap(pd.crosstab(df['work_year'],df['company_size']))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1e60cefd0>
```



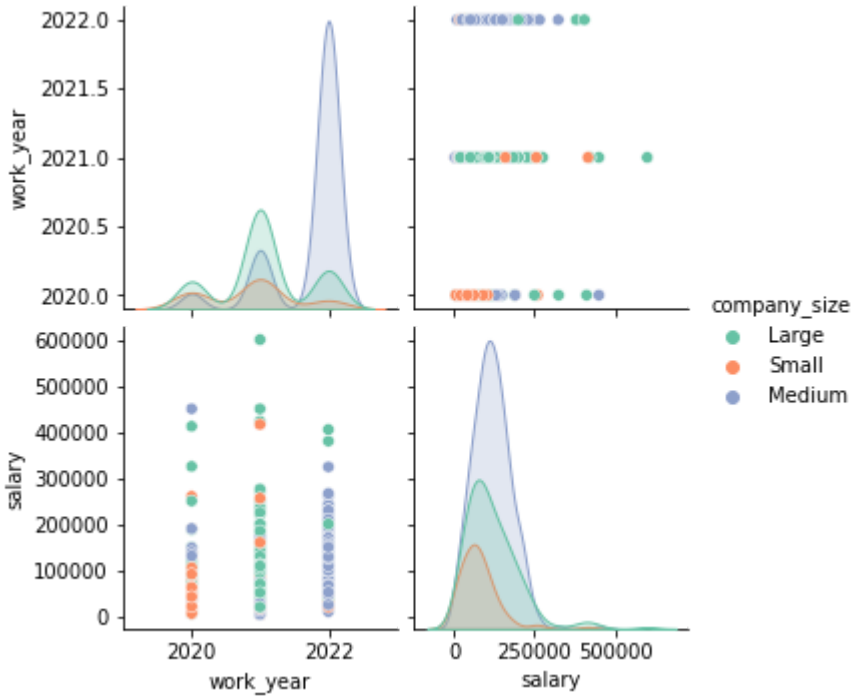

```
sns.clustermap(pd.crosstab(df['employment_type'],df['salary']))
```

```
<seaborn.matrix.ClusterGrid at 0x7fd1e5ec8c10>
```




```
sns.pairplot(df,hue='company_size')
```

```
<seaborn.axisgrid.PairGrid at 0x7fd1e6deb690>
```




```
# Building the model with 3 clusters
kmode = KModes(n_clusters=3, init = "random", n_init = 5, verbose=1)
clusters = kmode.fit_predict(df[['experience_level', 'salary','job_type']])
clusters
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 71, cost: 997.0
Run 1, iteration: 2/100, moves: 6, cost: 997.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 41, cost: 857.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 0, cost: 894.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 0, cost: 861.0
Init: initializing centroids
Init: initializing clusters
```

```
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 915.0
Best run was number 2
array([1, 2, 0, 1, 0, 0, 0, 1, 1, 0, 2, 1, 2, 1, 1, 1, 0, 0, 0, 1, 1, 1,
       0, 1, 1, 0, 0, 2, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 1, 2, 0, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 0, 0, 0, 0,
       2, 0, 0, 2, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 2, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 2, 0, 1, 2, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 2, 0, 1, 2, 2, 1, 0, 0, 1, 1, 0, 0, 2, 1, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 2, 1, 0, 0, 0, 0, 2, 0, 1, 1, 1, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 1, 0, 2, 1, 2, 1, 1, 1, 2, 0, 0, 0, 1, 1, 0,
       1, 1, 1, 1, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 2, 1, 0, 1, 2,
       1, 0, 1, 1, 1, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 1,
       1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0,
       0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 1, 0, 0, 0, 2, 2, 1,
       1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 1, 2, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 2, 2, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 2, 2, 1, 0, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
       1, 1, 1, 1, 2, 1, 0, 0, 0, 0, 0, 1, 0, 1, 2, 1, 0, 2, 1, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 1, 0, 0, 2, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       2, 2, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 2, 2, 0, 0, 2, 0, 1], dtype=uint16)
```

```
df.insert(0, "Cluster", clusters, True)
df
```

	Cluster	work_year	experience_level	employment_type	job_title	salary	employee
0	1	2020	Mid	Full-time	Data Scientist	79833	

```
df['Cluster'].value_counts()
```

```
0    326
1    209
2     72
Name: Cluster, dtype: int64
```

```
df = pd.read_csv('ds_salaries.csv')
df = df.drop(columns='Unnamed: 0')
df = df.drop(columns='experience_level')
df = df.drop(columns='employment_type')
df = df.drop(columns='job_title')
df = df.drop(columns= 'salary')
df = df.drop(columns='salary_currency')
df = df.drop(columns='employee_residence')
df = df.drop(columns='remote_ratio')
df = df.drop(columns= 'company_location')
df = df.drop(columns='company_size')
df.head()
```

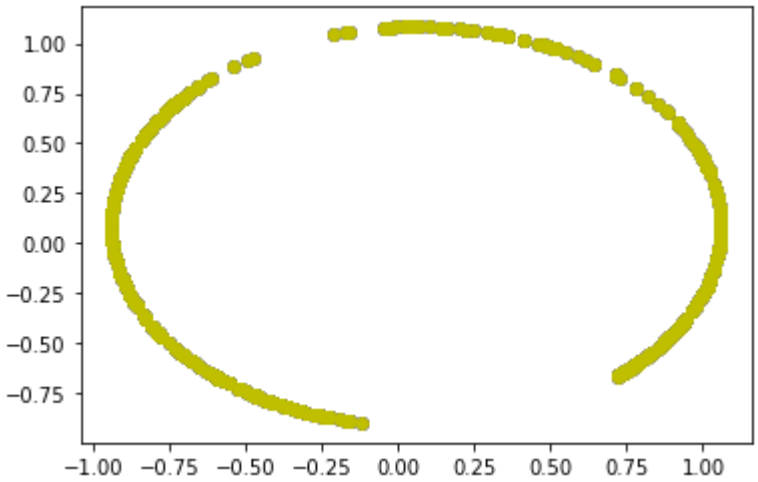
	work_year	salary_in_usd
0	2020	79833
1	2020	260000
2	2020	109024
3	2020	20000
4	2020	150000

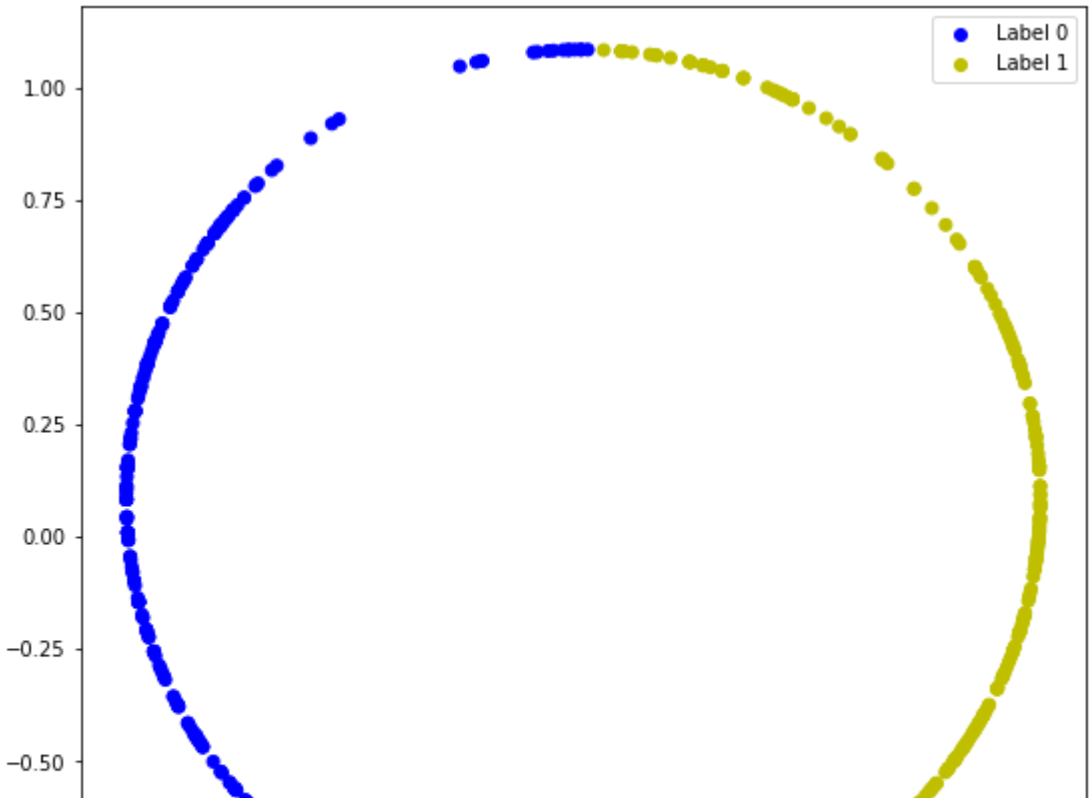
```
# Preprocessing the data to make it visualizable

# Scaling the Data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df)

# Normalizing the Data
X_normalized = normalize(X_scaled)

# Converting the numpy array into a pandas DataFrame
X_normalized = pd.DataFrame(X_normalized)

# Reducing the dimensions of the data
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(X_normalized)
```

```
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']

X_principal.head()
```

	P1	P2
0	1.049570	0.220670
1	0.461470	0.999480
2	1.003042	0.413736
3	1.036743	-0.125473
4	0.875933	0.660724

```
# Building the clustering model
spectral_model_rbf = SpectralClustering(n_clusters = 2, affinity ='rbf')

# Training the model and Storing the predicted cluster labels
labels_rbf = spectral_model_rbf.fit_predict(X_principal)



# Building the label to colour mapping
colours = {}
colours[0] = 'b'
colours[1] = 'y'
colours[2] = 'r'

# Building the colour vector for each data point
cvec = [colours[label] for label in labels_rbf]

# Plotting the clustered scatter plot

b = plt.scatter(X_principal['P1'], X_principal['P2'], color ='b');
y = plt.scatter(X_principal['P1'], X_principal['P2'], color ='y');

plt.figure(figsize =(9, 9))
plt.scatter(X_principal['P1'], X_principal['P2'], c = cvec)
plt.legend((b, y), ('Label 0', 'Label 1'))
plt.show()
```






```
# Building the clustering model
spectral_model_nn = SpectralClustering(n_clusters = 2, affinity ='nearest_neighbors')

# Training the model and Storing the predicted cluster labels
labels_nn = spectral_model_nn.fit_predict(X_principal)



# List of different values of affinity
affinity = ['rbf', 'nearest-neighbours']

# List of Silhouette Scores
s_scores = []

# Evaluating the performance
s_scores.append(silhouette_score(df, labels_rbf))
```

```
s_scores.append(silhouette_score(df, labels_nn))

print(s_scores)
```

```
[0.08340012329153063, 0.07095365013255728]
```