Term Project Data Mining - Gender Pay Gap Analysis

Milestone 1 - Week 6

Create a Graphical Analysis creating a minium of four grouphs. Label your graphs appropriately and explain/analyze provided by each graph. Your analysis should begin to answer the question(s) you are addressing. Write a short overview/conclusion of the insights gained from your graphical anaylsis.

In [1]:
```
# import the data set using necesarry libraries
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

# read csv Glassdoor Gender Pay Gap
df_pay = pd.read_csv("Glassdoor Gender Pay Gap.csv")
df_pay.head()
```

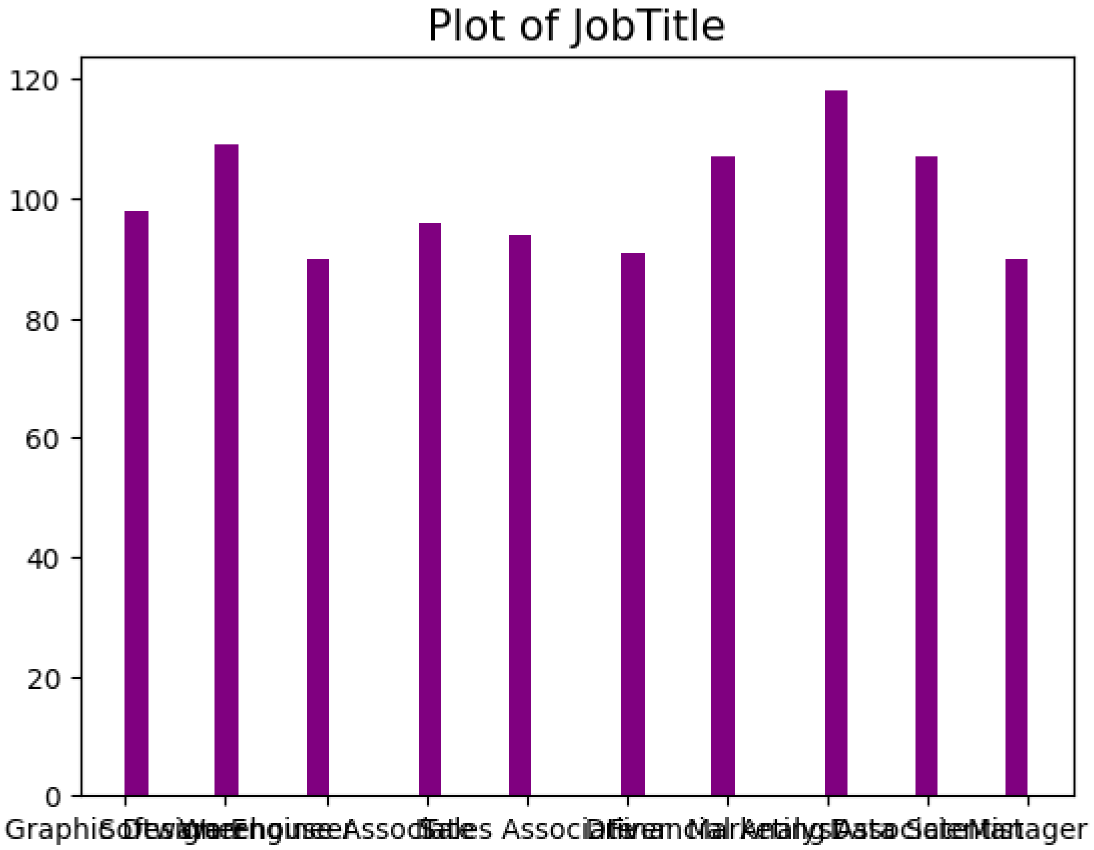
Out[1]:

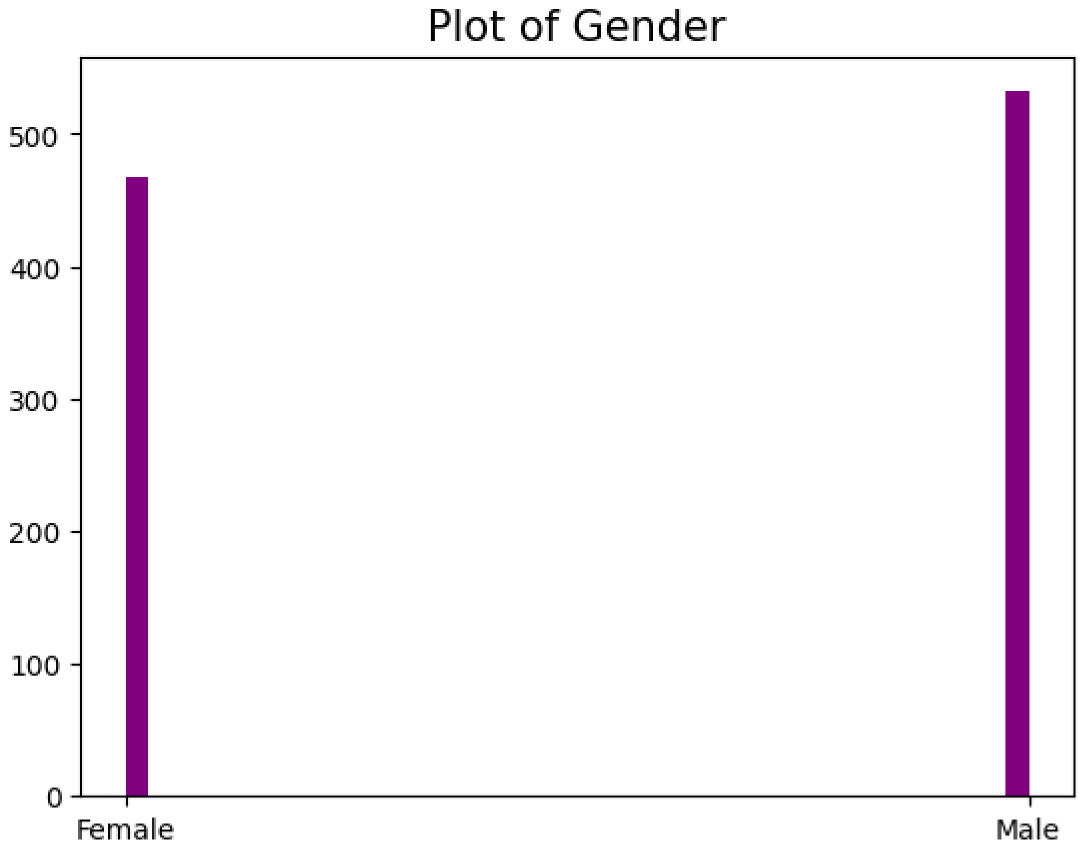
	JobTitle	Gender	Age	PerfEval	Education	Dept	Seniority	BasePay	Bonus
0	Graphic Designer	Female	18	5	College	Operations	2	42363	9938
1	Software Engineer	Male	21	5	College	Management	5	108476	11128
2	Warehouse Associate	Female	19	4	PhD	Administration	5	90208	9268
3	Software Engineer	Male	20	5	Masters	Sales	4	108080	10154
4	Graphic Designer	Male	26	5	Masters	Engineering	5	99464	9319

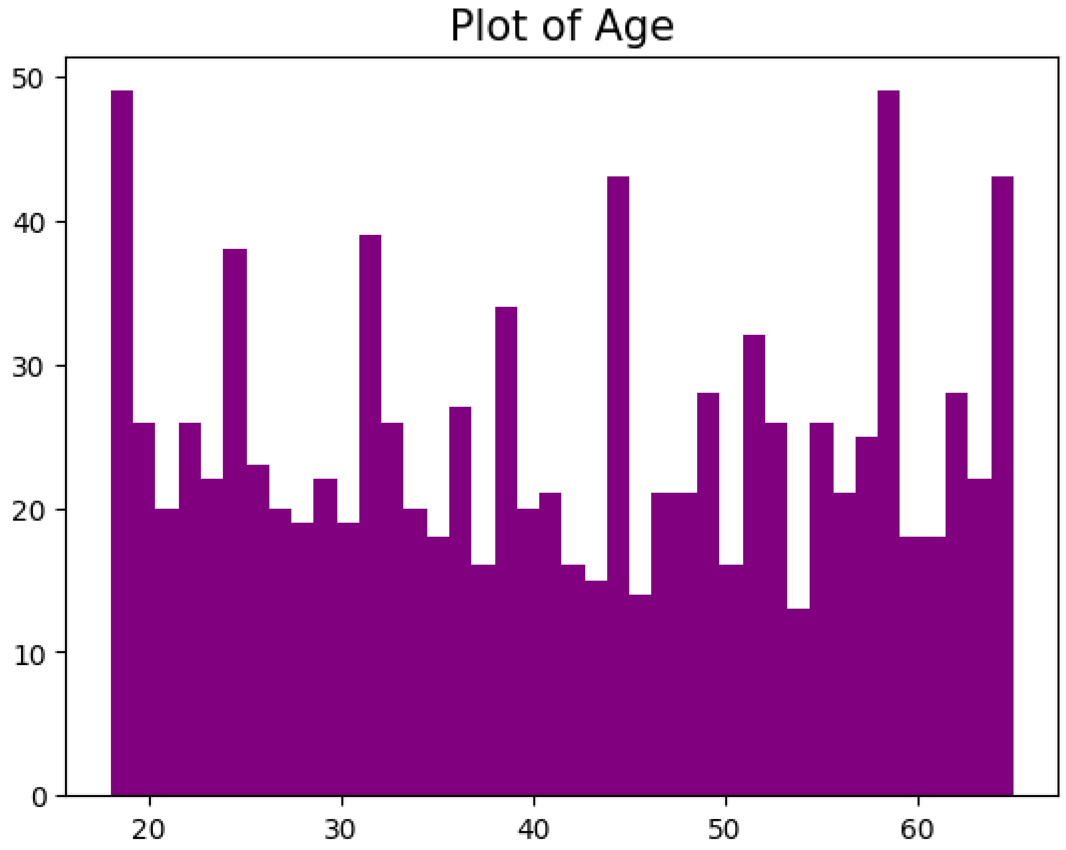
In [2]:
```
# find total number of records in csv by (rows, columns)
df_pay.shape
```

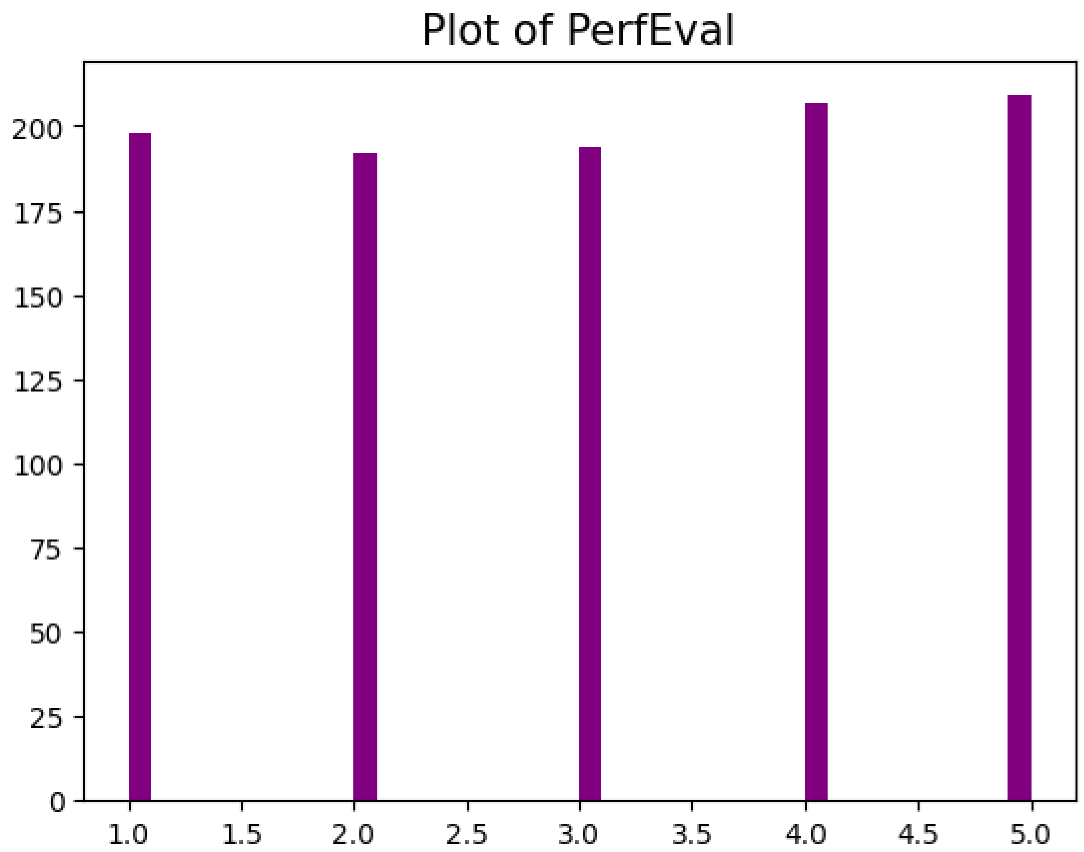
Out[2]:
```
(1000, 9)
```

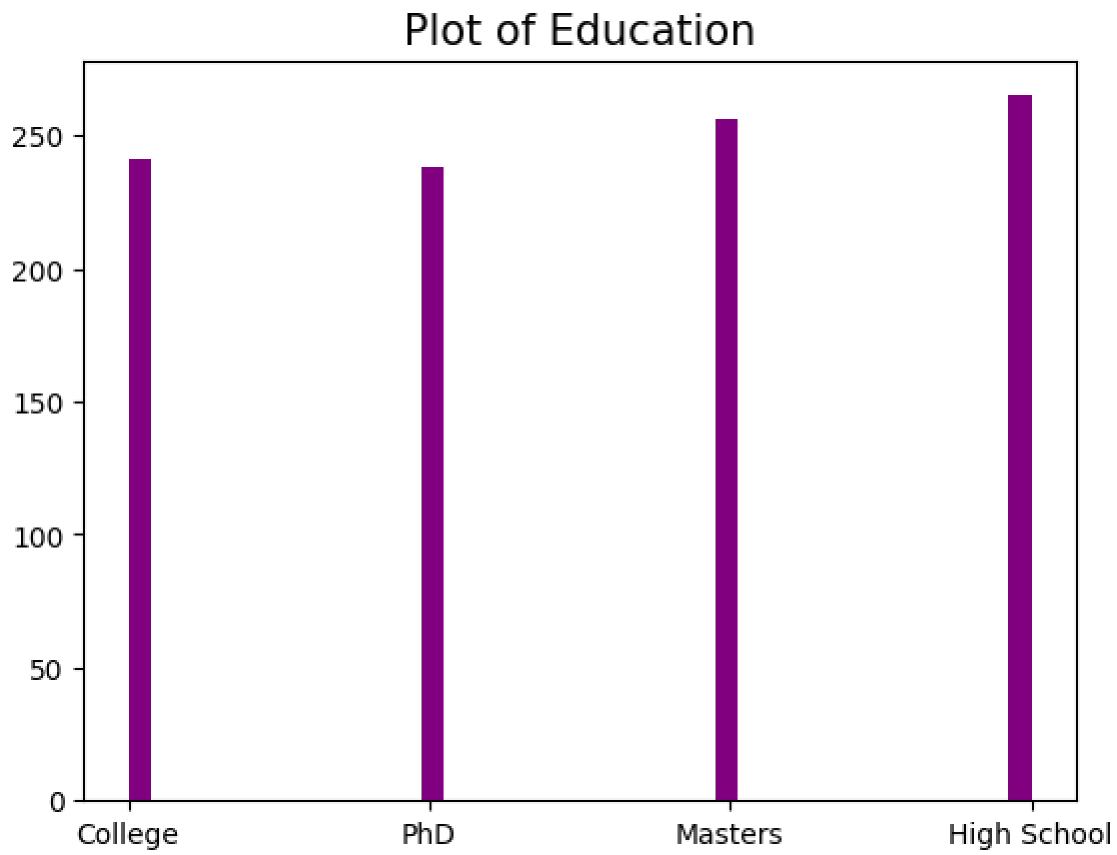
In [3]:
```
for c in df_pay.columns:
    plt.title("Plot of "+c,fontsize=15)
    plt.hist(df_pay[c],bins=40,color='purple')
    plt.show()
```

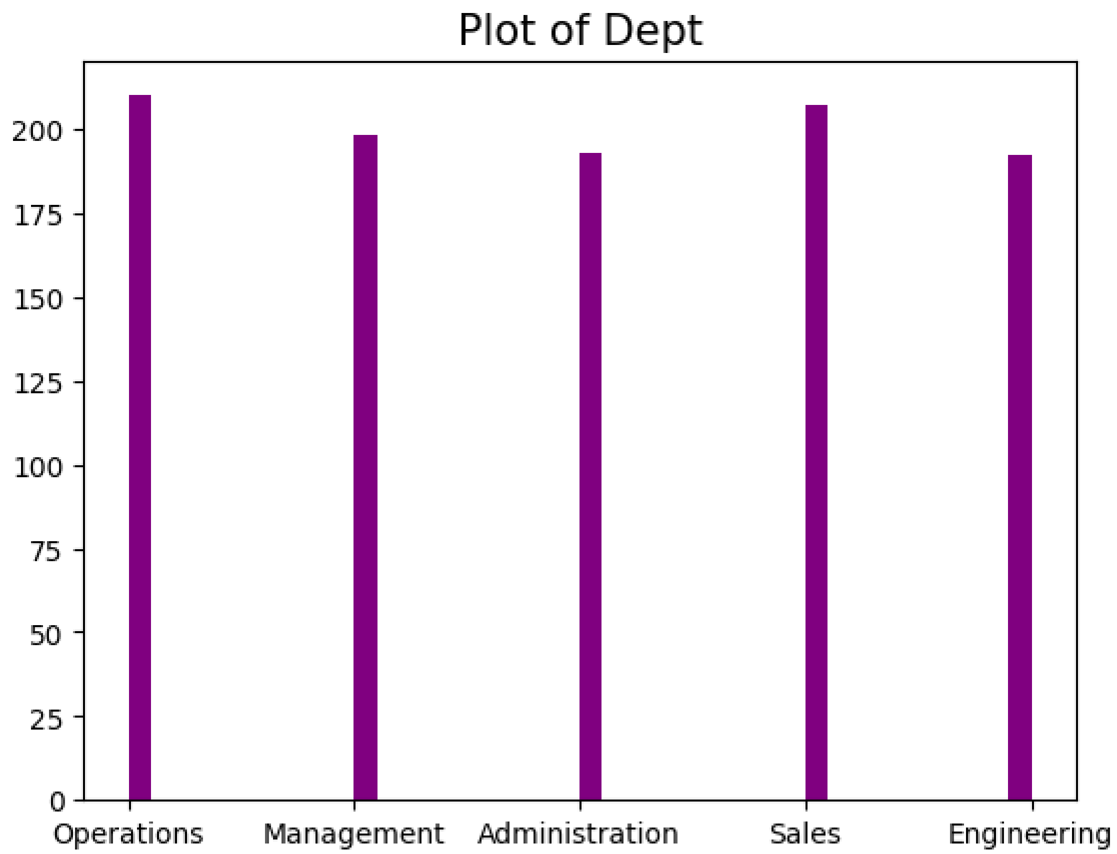

Plot of JobTitle
0
20
40
60
80
100
120


Plot of Gender
0
100
200
300
400
500
Female
Male

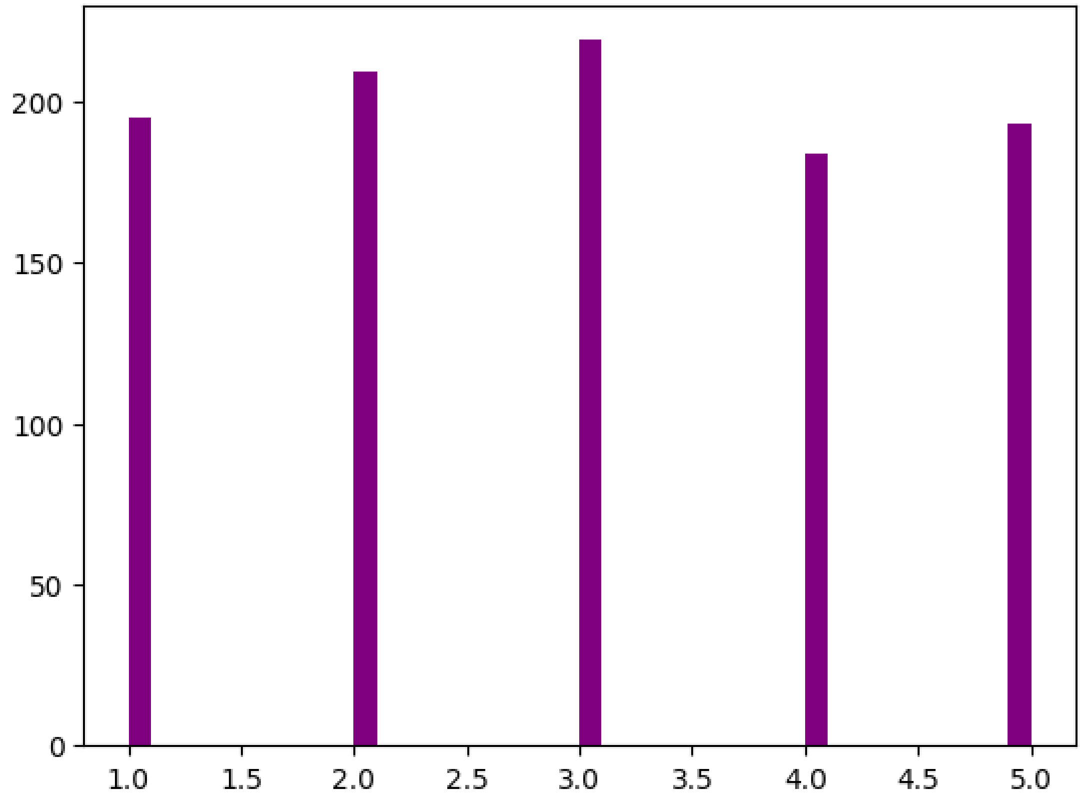

Plot of Age
0
10
20
30
40
50
20
30
40
50
60


Plot of PerfEval
0
25
50
75
100
125
150
175
200
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

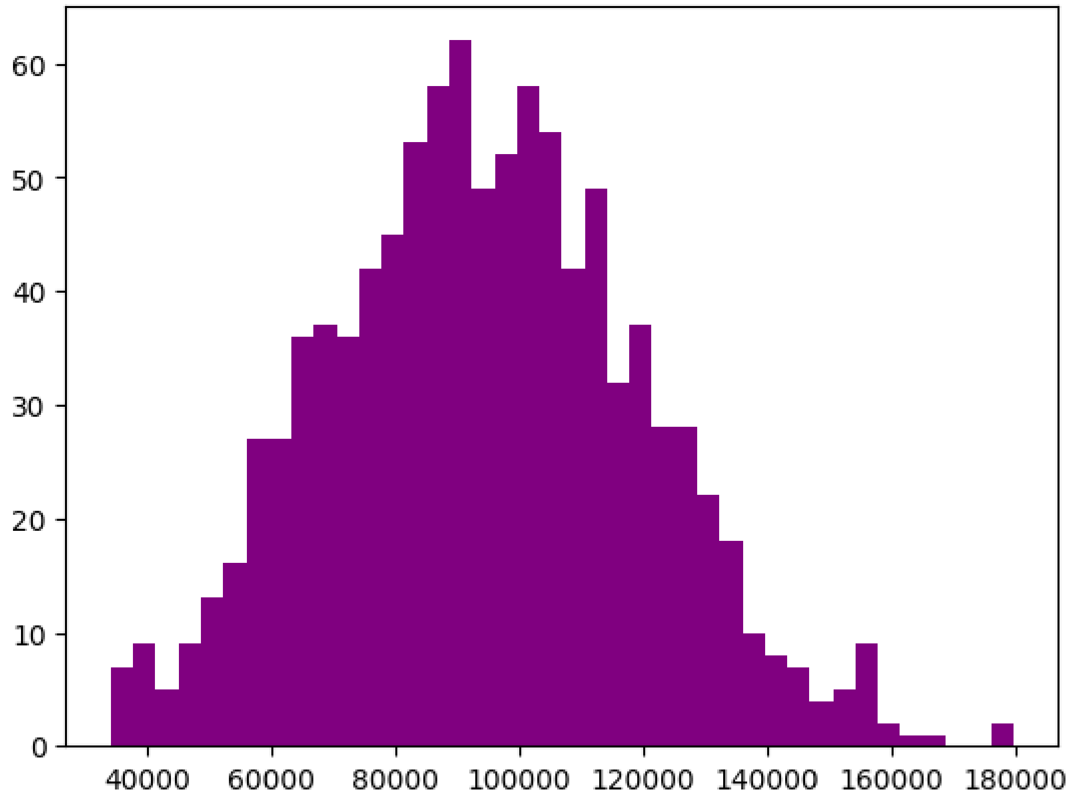

Plot of Education
250
200
150
100
50
0
College
PhD
Masters
High School

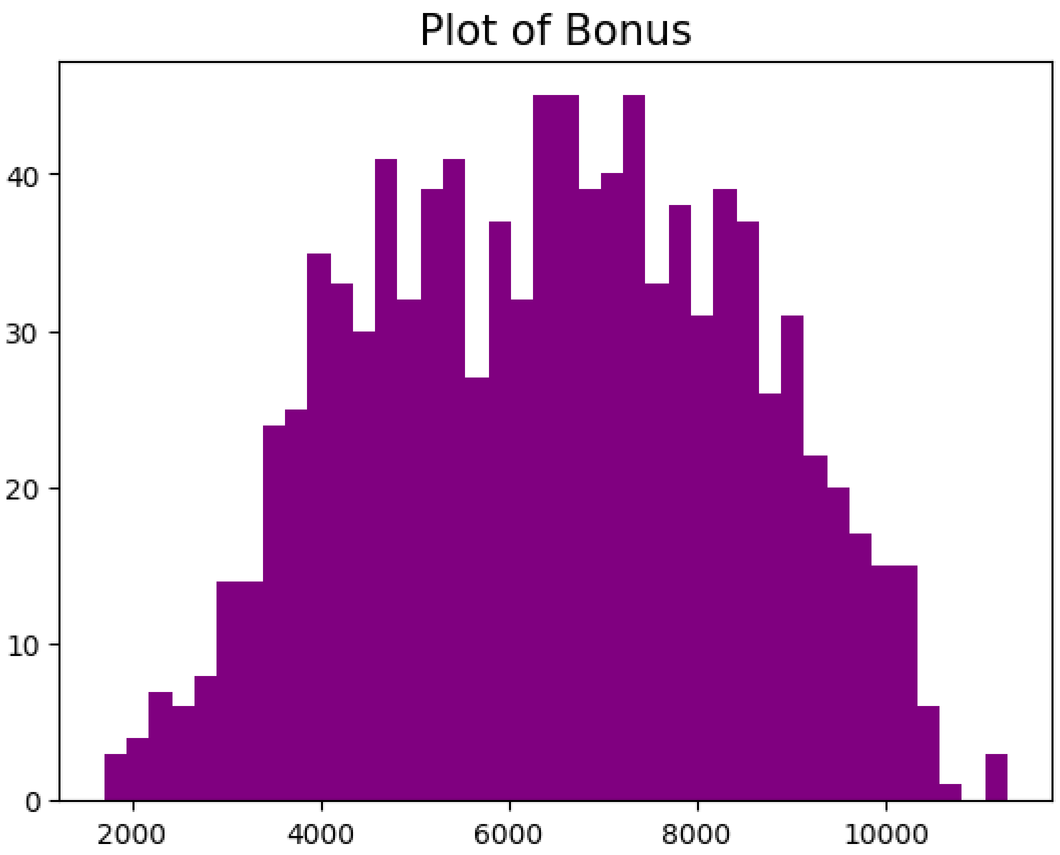

Plot of Dept
200
175
150
125
100
75
50
25
0
Operations
Management
Administration
Sales
Engineering

Plot of Seniority



Plot of BasePay

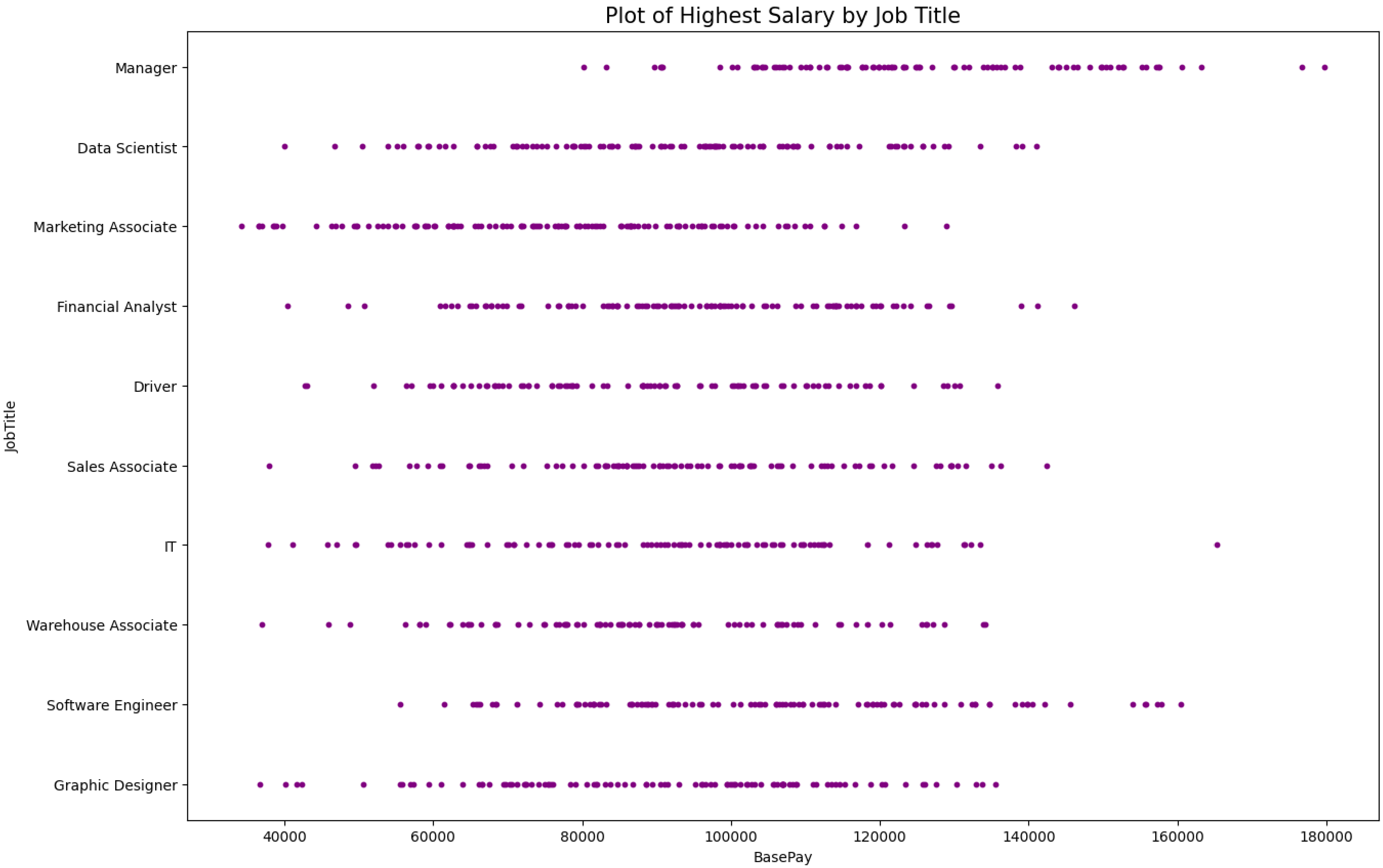




Which job title had the highest salary?

In [4]:
```
plt.figure(figsize =(15,10))
plt.title('Plot of Highest Salary by Job Title',fontsize=15)
plt.scatter(df_pay['BasePay'], df_pay['JobTitle'], s=10, color ='purple')
plt.xlabel('BasePay')
plt.ylabel('JobTitle')
plt.plot()
```

Out[4]: []



From the scatter plot above, I can see that the Highest Salary is of a Manager that is ranging around 180,000.

Which job had the highest bonus? Was it the same title as the highest salary?

In [5]:
```
# CORRECTED - changed from histogram to scatter plot to show visualization better.
plt.figure(figsize =(25,15))
plt.title('Plot of Highest Bonus by Job Title',fontsize=20)
```

```
plt.scatter(df_pay['Bonus'], df_pay['JobTitle'], s=15, color ='purple')
plt.xlabel('Bonus')
plt.ylabel('JobTitle')
plt.show()
```

CORRECTED -From the scatterplot above, it looks as thought the title that has the highest bonus is the Software Engineer with a bonus of 11,000 that a Manager shows.

Out of the highest salary and bonus, which gender reflected that salary?

In [6]:
```
# CORRECTED - changed from histogram to scatter plot to show visualization better.
plt.figure(figsize =(15,10))
```

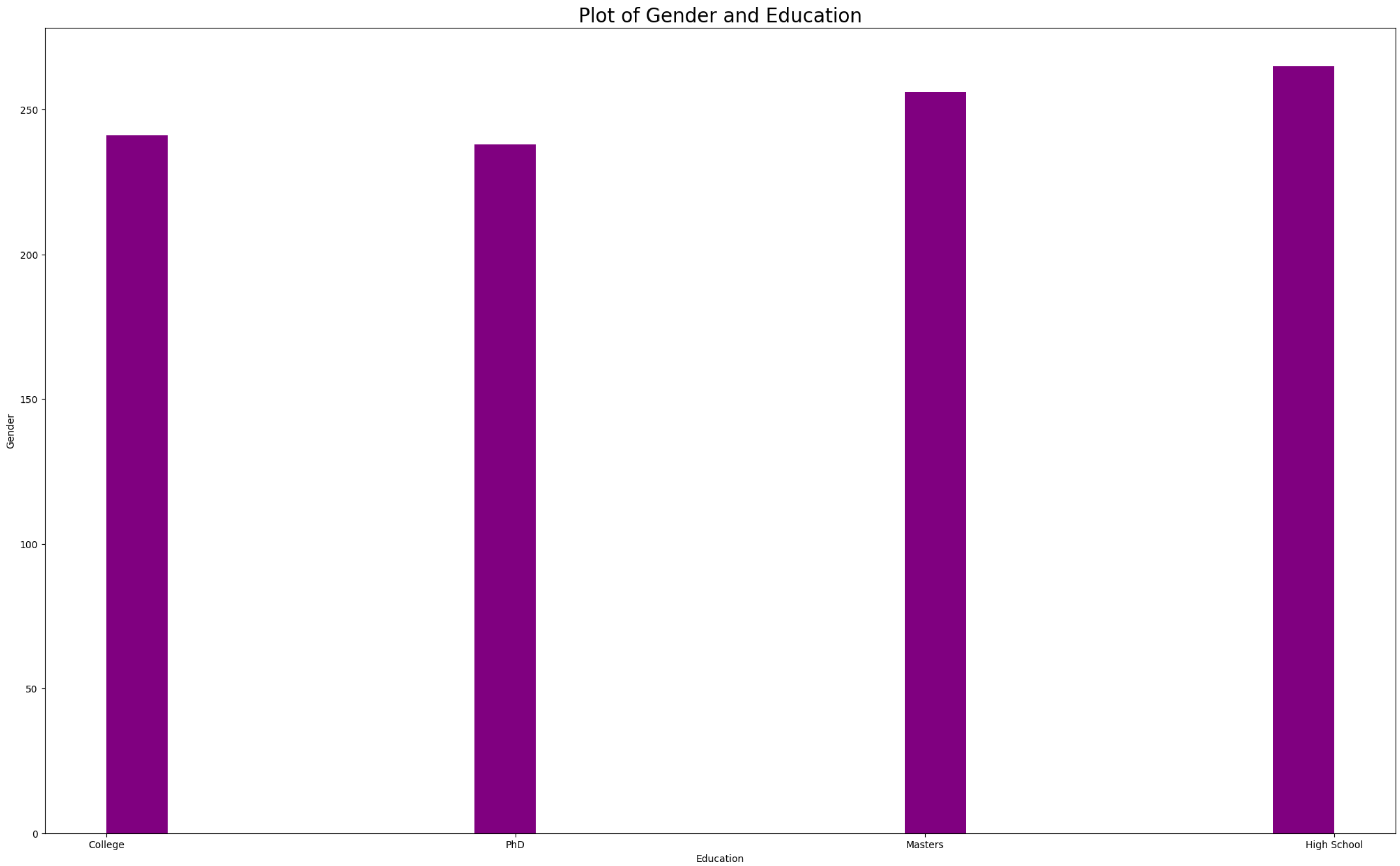
```
plt.title('Plot of Gender by Job Title: Manager',fontsize=15)
plt.scatter(df_pay['Gender'], df_pay['JobTitle']=='Manager', s=10, color ='purple')
plt.xlabel('Gender')
plt.ylabel('JobTitle')
plt.plot()
```

Out[6]: []

CORRECTED -From the scatterplot above, it looks as though since the Manager had the highest salary, and that both genders where considered for the Manager position. Though the graph doesn't show exactly what gender reflected the salary of 180,000.

Did the opposite gender have the same schooling as the gender that had the highest salary?

In [7]:
```
plt.figure(figsize =(25,15))
plt.title('Plot of Gender and Education',fontsize=20)
plt.hist(df_pay['Education'],bins=20,color='purple')
plt.xlabel('Education')
plt.ylabel('Gender')
plt.show()
```



CORRECTED- I was able to correct all of my graphs to show more of what I was looking for with my questions. However, With the last one, it does show the range for the education degrees, but still couldn't figure out how to pull which gender had which education using matplotlib as my visualization source.

DSC550-T30 Term Project: Milestone 2

8.2: Data Preparation for CSV File

Step 1. Import Libraries and Load Files

In [1]:
```
# Load the necessary libraries
# import pandas and numpy libraries
import pandas as pd
import numpy as np
```

Step 2. Read the csv file

In [2]:
```
# read csv Glassdoor Gender Pay Gap
df_pay = pd.read_csv("Glassdoor Gender Pay Gap.csv")
# check information that is on the file by using .head function
df_pay.head()
```

Out[2]:

	JobTitle	Gender	Age	PerfEval	Education	Dept	Seniority	BasePay	Bonus
0	Graphic Designer	Female	18	5	College	Operations	2	42363	9938
1	Software Engineer	Male	21	5	College	Management	5	108476	11128
2	Warehouse Associate	Female	19	4	PhD	Administration	5	90208	9268
3	Software Engineer	Male	20	5	Masters	Sales	4	108080	10154
4	Graphic Designer	Male	26	5	Masters	Engineering	5	99464	9319

In [3]:
```
# check to see how many (rows, columns) by using shape function
df_pay.shape
```

Out[3]: (1000, 9)

Step 3. Check for missing data for csv file

In [4]:
```
## checking to see if there is any missing data in csv file
pd.isnull(df_pay).head()
```

Out[4]:

	JobTitle	Gender	Age	PerfEval	Education	Dept	Seniority	BasePay	Bonus
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False

In [5]:
```
## selecting only the rows with NaN values using any function
df_pay[pd.isnull(df_pay).any(axis=1)]
```

Out[5]:

JobTitle	Gender	Age	PerfEval	Education	Dept	Seniority	BasePay	Bonus

In [6]:
```
# drop rows that are all NA using drop NA function
df_pay.dropna(how='all')
```

Out[6]:

	JobTitle	Gender	Age	PerfEval	Education	Dept	Seniority	BasePay	Bonus
0	Graphic Designer	Female	18	5	College	Operations	2	42363	9938
1	Software Engineer	Male	21	5	College	Management	5	108476	11128
2	Warehouse Associate	Female	19	4	PhD	Administration	5	90208	9268
3	Software Engineer	Male	20	5	Masters	Sales	4	108080	10154
4	Graphic Designer	Male	26	5	Masters	Engineering	5	99464	9319
...	...	...	...	...	...	...	...	...	...
995	Marketing Associate	Female	61	1	High School	Administration	1	62644	3270
996	Data Scientist	Male	57	1	Masters	Sales	2	108977	3567
997	Financial Analyst	Male	48	1	High School	Operations	1	92347	2724
998	Financial Analyst	Male	65	2	High School	Administration	1	97376	2225
999	Financial Analyst	Male	60	1	PhD	Sales	2	123108	2244

1000 rows × 9 columns

In [7]:
```
# drop columns that are all NA using drop NA function
df_pay.dropna(axis =1, how='all')
```

Out[7]:

	JobTitle	Gender	Age	PerfEval	Education	Dept	Seniority	BasePay	Bonus
0	Graphic Designer	Female	18	5	College	Operations	2	42363	9938
1	Software Engineer	Male	21	5	College	Management	5	108476	11128
2	Warehouse Associate	Female	19	4	PhD	Administration	5	90208	9268
3	Software Engineer	Male	20	5	Masters	Sales	4	108080	10154
4	Graphic Designer	Male	26	5	Masters	Engineering	5	99464	9319
...	...	...	...	...	...	...	...	...	...
995	Marketing Associate	Female	61	1	High School	Administration	1	62644	3270
996	Data Scientist	Male	57	1	Masters	Sales	2	108977	3567
997	Financial Analyst	Male	48	1	High School	Operations	1	92347	2724
998	Financial Analyst	Male	65	2	High School	Administration	1	97376	2225
999	Financial Analyst	Male	60	1	PhD	Sales	2	123108	2244

1000 rows × 9 columns

Step 4. Filling missing data.

In [8]:
```
# using the fillna, isnull, and sum to fill in the NA values with non-null data
df_pay = df_pay.fillna(0)
print(df_pay.isnull().sum())
```

```
JobTitle      0
Gender        0
Age           0
PerfEval      0
Education     0
Dept          0
Seniority     0
BasePay       0
Bonus         0
dtype: int64
```

Step 5. Remove duplicates from csv files

In [9]:
```
## removing duplicates from rows and printing out the results
duplicate = df_pay[df_pay.duplicated()]
print("\n\nDuplicate Rows : \n {}".format(duplicate))
drop_duplicate = df_pay.drop_duplicates(keep=False)
print("\n\nResult of CSV after duplicates are removed :\n",drop_duplicate.head())
```

```


Duplicate Rows :
 Empty DataFrame
Columns: [JobTitle, Gender, Age, PerfEval, Education, Dept, Seniority, BasePay, Bonus]
Index: []


Result of CSV after duplicates are removed :
               JobTitle  Gender  Age  PerfEval Education            Dept  \
0     Graphic Designer  Female   18         5   College      Operations
1    Software Engineer    Male   21         5   College      Management
2  Warehouse Associate  Female   19         4       PhD  Administration
3    Software Engineer    Male   20         5   Masters           Sales
4     Graphic Designer    Male   26         5   Masters     Engineering

   Seniority  BasePay  Bonus
0          2    42363   9938
1          5   108476  11128
2          5    90208   9268
3          4   108080  10154
4          5    99464   9319
```

Step 6. Rename the columns

In [10]:
```
## checking to see what the column names are
df_pay.columns
```

Out[10]:
```
Index(['JobTitle', 'Gender', 'Age', 'PerfEval', 'Education', 'Dept',
       'Seniority', 'BasePay', 'Bonus'],
      dtype='object')
```

In [11]:
```
## create new dataframe to rename column names using rename method
df_pay = df_pay.rename(columns = {'JobTitle':'Job_Title', 'Gender':'Gender', 'Age':'Age', 'PerfEval':'Perf_Eval',
'Education':'Education', 'Dept':'Dept',
'Seniority':'Seniority', 'BasePay':'Base_Pay',
```

```
'Bonus':'Bonus'}, inplace = False)
df_pay.head()
```

Out[11]:

	Job_Title	Gender	Age	Perf_Eval	Education	Dept	Seniority	Base_Pay	Bonus
0	Graphic Designer	Female	18	5	College	Operations	2	42363	9938
1	Software Engineer	Male	21	5	College	Management	5	108476	11128
2	Warehouse Associate	Female	19	4	PhD	Administration	5	90208	9268
3	Software Engineer	Male	20	5	Masters	Sales	4	108080	10154
4	Graphic Designer	Male	26	5	Masters	Engineering	5	99464	9319

Step 7. Create new csv file

In [12]:
```
# after renaming columns and filling in missing values write a new csv file
df_pay.to_csv('Gender_Pay_Gap.csv')
```

In [13]:
```
# check to see how many (rows, columns) by using shape function for new csv
df_pay.shape
```

Out[13]:
```
(1000, 9)
```

DSC550-T301 Term Project: Milestone 3

10.2 Model Building and Evaluation

Step. 1 Import Libraries and Load Files

In [1]:
```
# load all the necessary libraries
# import pandas, numpy, and sklearn models
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

Step 2. Load cleaned CSV file

In [2]:
```
# read csv Glassdoor Gender Pay Gap
df_pay = pd.read_csv("Gender_Pay_Gap.csv")
# check information that is on the file by using .head function
df_pay.head()
```

Out[2]:

	Unnamed: 0	Job_Title	Gender	Age	Perf_Eval	Education	Dept	Seniority	Base_Pay	Bonus
0	0	Graphic Designer	Female	18	5	College	Operations	2	42363	9938
1	1	Software Engineer	Male	21	5	College	Management	5	108476	11128
2	2	Warehouse Associate	Female	19	4	PhD	Administration	5	90208	9268
3	3	Software Engineer	Male	20	5	Masters	Sales	4	108080	10154
4	4	Graphic Designer	Male	26	5	Masters	Engineering	5	99464	9319

In [3]:
```
# check to see how many (rows, columns) by using shape function
df_pay.shape
```

Out[3]:
```
(1000, 10)
```

In [4]:
```
# statistical summary of each attribute
print(df_pay.describe())
```

```
        Unnamed: 0          Age    Perf_Eval    Seniority       Base_Pay  \
count  1000.000000  1000.000000  1000.000000  1000.000000    1000.000000
mean    499.500000    41.393000     3.037000     2.971000   94472.653000
std     288.819436    14.294856     1.423959     1.395029   25337.493272
min       0.000000    18.000000     1.000000     1.000000   34208.000000
25%     249.750000    29.000000     2.000000     2.000000   76850.250000
50%     499.500000    41.000000     3.000000     3.000000   93327.500000
75%     749.250000    54.250000     4.000000     4.000000  111558.000000
max     999.000000    65.000000     5.000000     5.000000  179726.000000

              Bonus
count   1000.000000
mean    6467.161000
std     2004.377365
min     1703.000000
25%     4849.500000
50%     6507.000000
75%     8026.000000
max    11293.000000
```

Step 4. Class Distribution

In [5]:
```
# using groupby function to see class distribution by age
print(df_pay.groupby('Age').size())
```

```
Age
18    29
19    20
20    26
21    20
22    26
23    22
24    24
25    14
26    23
27    20
28    19
29    22
30    19
31    22
32    17
33    26
34    20
35    18
36    27
37    16
38    17
39    17
40    20
41    21
42    16
43    15
44    18
45    25
46    14
47    21
48    21
49    28
50    16
51    18
52    14
53    26
54    13
55    26
56    21
57    25
58    26
59    23
60    18
61    18
62    28
63    22
64    21
65    22
dtype: int64
```

In [6]:
```
# using groupby function to see class distribution by base pay
print(df_pay.groupby('Base_Pay').size())
```

```
Base_Pay
34208     1
36548     1
36585     1
36642     1
36972     1
         ..
160614    1
163208    1
165229    1
176789    1
179726    1
Length: 992, dtype: int64
```

Step 5. Create a Validation using train_test_split

In [7]:
```
# split out validation train
array = df_pay.values
x = array[:,0:10]
y = array[:,9]
# use sklearn to split data
from sklearn.model_selection import train_test_split
X_train, X_validation, Y_train, Y_validation = train_test_split(x, y, test_size=0.3, random_state=42)
```

Step 6. Build Models, Evaluate and Compare

In [8]:
```
# build models from sklearn
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
```

In [9]:
```
# evaluate each model from sklearn
results = []
names = []
for name,model in models:
    kfold = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
print('%s: %f (%f)' % (model, cv_results.mean(), cv_results.std()))
```

```
---------------------------------------------------------------------------
Empty                                     Traceback (most recent call last)
File ~\PycharmProjects\pythonProject\venv\lib\site-packages\joblib\parallel.py:862, in Parallel.dispatch_one_batch(self, iterator)
    861 try:
--> 862     tasks = self._ready_batches.get(block=False)
    863 except queue.Empty:
    864     # slice the iterator n_jobs * batchsize items at a time. If the
    865     # slice returns less than that, then the current batchsize puts
   (...)
    868     # accordingly to distribute evenly the last items between all
    869     # workers.

File ~\AppData\Local\Programs\Python\Python310\lib\queue.py:168, in Queue.get(self, block, timeout)
    167     if not self._qsize():
--> 168         raise Empty
    169 elif timeout is None:

Empty: 

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
Cell In[9], line 6
      4 for name,model in models:
      5     kfold = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)
----> 6     cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
      7     results.append(cv_results)
      8     names.append(name)

File ~\PycharmProjects\pythonProject\venv\lib\site-packages\sklearn\model_selection\_validation.py:515, in cross_val_score(estimator, X, y, groups, scoring, cv, n_jobs, verbose, fit_params, pre_dispatch, error_score)
    512 # To ensure multimetric format is not supported
    513 scorer = check_scoring(estimator, scoring=scoring)
--> 515 cv_results = cross_validate(
    516     estimator=estimator,
    517     X=X,
    518     y=y,
    519     groups=groups,
    520     scoring={"score": scorer},
    521     cv=cv,
    522     n_jobs=n_jobs,
    523     verbose=verbose,
    524     fit_params=fit_params,
    525     pre_dispatch=pre_dispatch,
    526     error_score=error_score,
    527 )
    528 return cv_results["test_score"]

File ~\PycharmProjects\pythonProject\venv\lib\site-packages\sklearn\model_selection\_validation.py:266, in cross_validate(estimator, X, y, groups, scoring, cv, n_jobs, verbose, fit_params, pre_dispatch, return_train_score, return_estimator, error_score)
    263 # We clone the estimator to make sure that all the folds are
    264 # independent, and that it is pickle-able.
    265 parallel = Parallel(n_jobs=n_jobs, verbose=verbose, pre_dispatch=pre_dispatch)
--> 266 results = parallel(
    267     delayed(_fit_and_score)(
    268         clone(estimator),
    269         X,
    270         y,
    271         scorers,
    272         train,
```

```
    273         test,
    274         verbose,
    275         None,
    276         fit_params,
    277         return_train_score=return_train_score,
    278         return_times=True,
    279         return_estimator=return_estimator,
    280         error_score=error_score,
    281     )
    282     for train, test in cv.split(X, y, groups)
    283 )
    285 _warn_or_raise_about_fit_failures(results, error_score)
    287 # For callabe scoring, the return type is only know after calling. If the
    288 # return type is a dictionary, the error scores can now be inserted with
    289 # the correct key.

File ~\PycharmProjects\pythonProject\venv\lib\site-packages\joblib\parallel.py:1085, in Parallel.__call__(self, iterable)
   1076 try:
   1077     # Only set self._iterating to True if at least a batch
   1078     # was dispatched. In particular this covers the edge
   (...)
   1082     # was very quick and its callback already dispatched all the
   1083     # remaining jobs.
   1084     self._iterating = False
-> 1085     if self.dispatch_one_batch(iterator):
   1086         self._iterating = self._original_iterator is not None
   1088     while self.dispatch_one_batch(iterator):

File ~\PycharmProjects\pythonProject\venv\lib\site-packages\joblib\parallel.py:873, in Parallel.dispatch_one_batch(self, iterator)
    870 n_jobs = self._cached_effective_n_jobs
    871 big_batch_size = batch_size * n_jobs
--> 873 islice = list(itertools.islice(iterator, big_batch_size))
    874 if len(islice) == 0:
    875     return False

File ~\PycharmProjects\pythonProject\venv\lib\site-packages\sklearn\model_selection\_validation.py:266, in <genexpr>(.0)
    263 # We clone the estimator to make sure that all the folds are
    264 # independent, and that it is pickle-able.
    265 parallel = Parallel(n_jobs=n_jobs, verbose=verbose, pre_dispatch=pre_dispatch)
--> 266 results = parallel(
    267     delayed(_fit_and_score)(
    268         clone(estimator),
    269         X,
    270         y,
    271         scorers,
    272         train,
    273         test,
    274         verbose,
    275         None,
    276         fit_params,
    277         return_train_score=return_train_score,
    278         return_times=True,
    279         return_estimator=return_estimator,
    280         error_score=error_score,
    281     )
    282     for train, test in cv.split(X, y, groups)
    283 )
    285 _warn_or_raise_about_fit_failures(results, error_score)
    287 # For callabe scoring, the return type is only know after calling. If the
    288 # return type is a dictionary, the error scores can now be inserted with
```

```
    289 # the correct key.

File ~\PycharmProjects\pythonProject\venv\lib\site-packages\sklearn\model_selection\_split.py:352, in _BaseKFold.split(self, X, y, groups)
    344 if self.n_splits > n_samples:
    345     raise ValueError(
    346         (
    347             "Cannot have number of splits n_splits={0} greater"
    348             " than the number of samples: n_samples={1}."
    349         ).format(self.n_splits, n_samples)
    350     )
--> 352 for train, test in super().split(X, y, groups):
    353     yield train, test

File ~\PycharmProjects\pythonProject\venv\lib\site-packages\sklearn\model_selection\_split.py:85, in BaseCrossValidator.split(self, X, y, groups)
     83 X, y, groups = indexable(X, y, groups)
     84 indices = np.arange(_num_samples(X))
---> 85 for test_index in self._iter_test_masks(X, y, groups):
     86     train_index = indices[np.logical_not(test_index)]
     87     test_index = indices[test_index]

File ~\PycharmProjects\pythonProject\venv\lib\site-packages\sklearn\model_selection\_split.py:733, in StratifiedKFold._iter_test_masks(self, X, y, groups)
    732 def _iter_test_masks(self, X, y=None, groups=None):
--> 733     test_folds = self._make_test_folds(X, y)
    734     for i in range(self.n_splits):
    735         yield test_folds == i

File ~\PycharmProjects\pythonProject\venv\lib\site-packages\sklearn\model_selection\_split.py:676, in StratifiedKFold._make_test_folds(self, X, y)
    674 allowed_target_types = ("binary", "multiclass")
    675 if type_of_target_y not in allowed_target_types:
--> 676     raise ValueError(
    677         "Supported target types are: {}. Got {!r} instead.".format(
    678             allowed_target_types, type_of_target_y
    679         )
    680     )
    682 y = column_or_1d(y)
    684 _, y_idx, y_inv = np.unique(y, return_index=True, return_inverse=True)

ValueError: Supported target types are: ('binary', 'multiclass'). Got 'unknown' instead.
```

In [ ]:
```
# create a comparison of models data using a boxplot
from matplotlib import pyplot as plt
plt.boxplot(results, labels=names)
plt.title('Model Evaluation Comparison')
plt.show()
```

Step 7. Make a prediction and evaluate

In [ ]:
```
# create a prediction from validation data
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
```

In [ ]:
```
# evaluate the predictions
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```