NB_hw_3

October 30, 2023

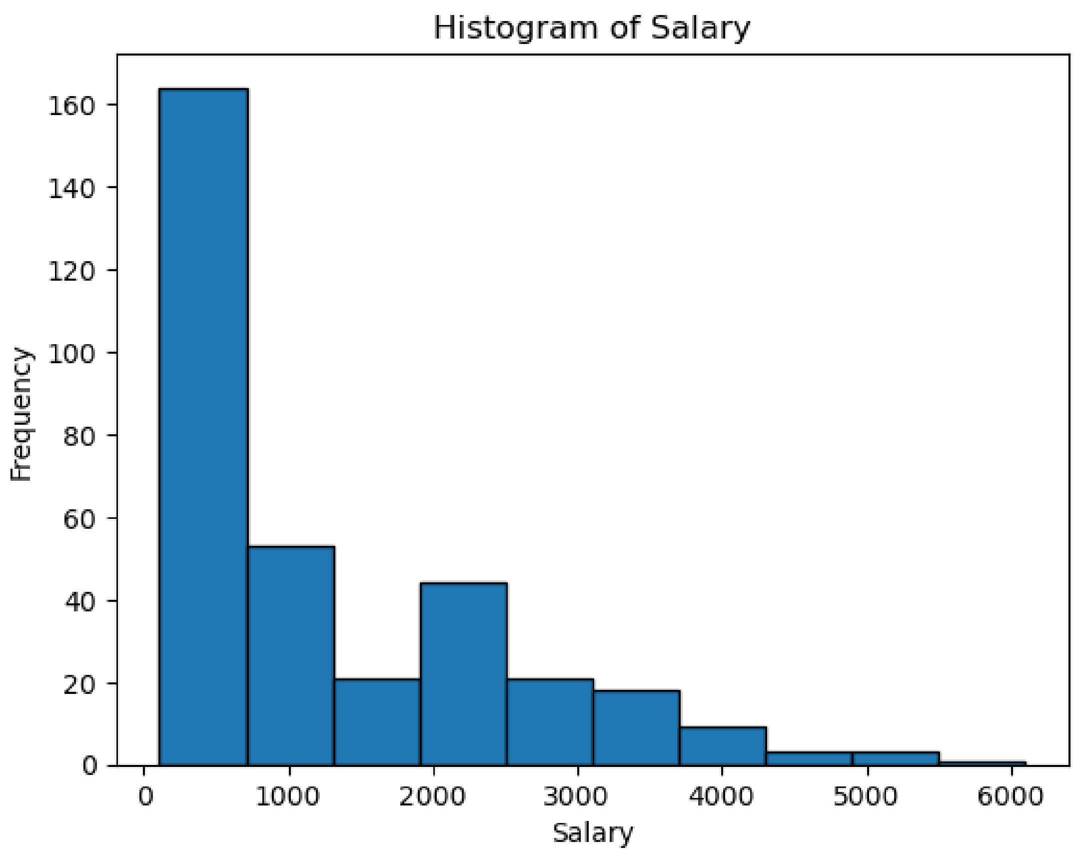
[78]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as smf
```

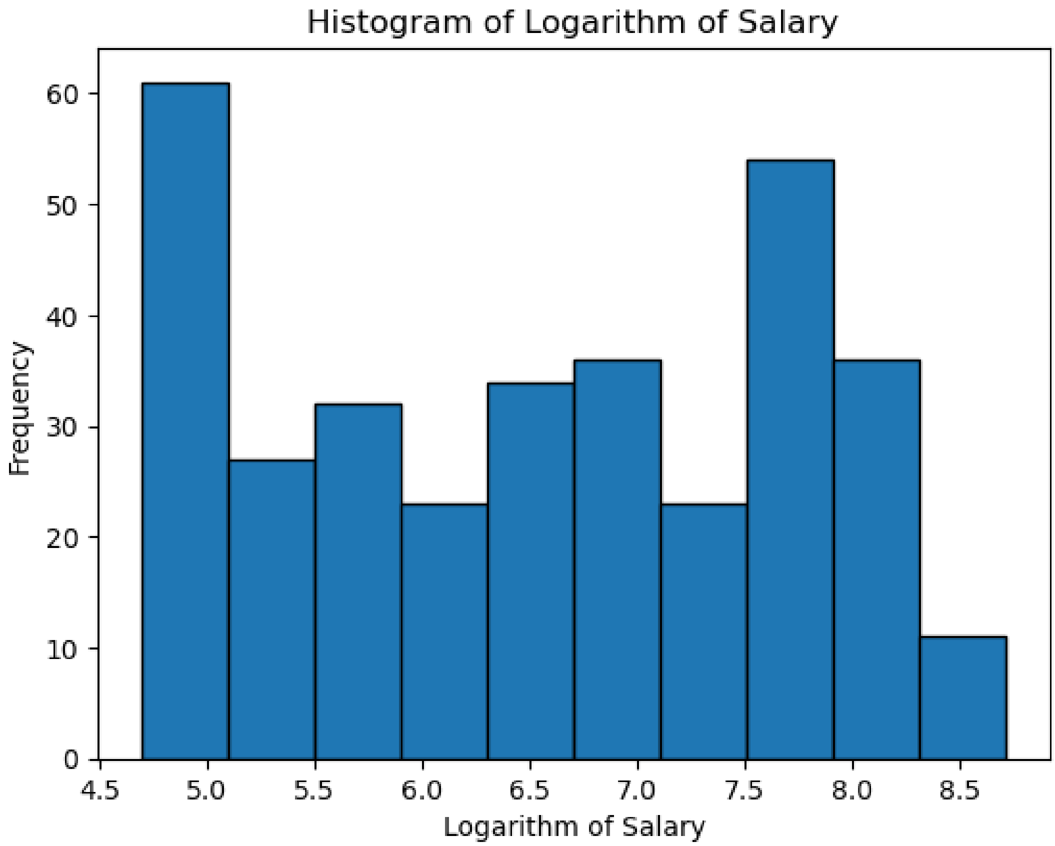
[79]:
```python
baseball = pd.read_table('http://jse.amstat.org/datasets/baseball.dat.txt',␣
↪header=None, sep="\s+", names=["salary", "batting.avg", "OBP", "runs",␣
↪"hits", "doubles", "triples", "homeruns", "RBI", "walks", "strike.outs",␣
↪"stolen.bases", "errors", "free.agency.elig", "free.agent.91", "arb.elig",␣
↪"arb.91", "name"])
baseball.head()
```

[79]:
```
   salary  batting.avg    OBP  runs  hits  doubles  triples  homeruns  RBI  \
0    3300        0.272  0.302    69   153       21        4        31  104
1    2600        0.269  0.335    58   111       17        2        18   66
2    2500        0.249  0.337    54   115       15        1        17   73
3    2475        0.260  0.292    59   128       22        7        12   50
4    2313        0.273  0.346    87   169       28        5         8   58

   walks  strike.outs  stolen.bases  errors  free.agency.elig  free.agent.91  \
0     22           80             4       3                 1              0
1     39           69             0       3                 1              1
2     63          116             6       5                 1              0
3     23           64            21      21                 0              0
4     70           53             3       8                 0              0

   arb.elig  arb.91             name
0         0       0  Andre Dawson
1         0       0  Steve Buchele
2         0       0  Kal Daniels
3         1       0  Shawon Dunston
4         1       0  Mark Grace
```

1 Exploratory Data Analysis: First prepare your data

[80]:
```python
#a
import matplotlib.pyplot as plt

# Histogram of salary
plt.hist(baseball['salary'], bins=10, edgecolor='black')
plt.xlabel('Salary')
plt.ylabel('Frequency')
plt.title('Histogram of Salary')
plt.show()

# Histogram of logarithm of salary
plt.hist(np.log(baseball['salary']), bins=10, edgecolor='black')
plt.xlabel('Logarithm of Salary')
plt.ylabel('Frequency')
plt.title('Histogram of Logarithm of Salary')
plt.show()
```





Before the logarithmic transformation, the salary data in the baseball dataset was highly right-skewed. This means that there were a few players with very high salaries, while the majority of players had lower salaries.

After applying the logarithmic transformation to the salary data, the distribution of salaries became more normally distributed: the salaries were spread out more evenly across the range of values, resulting in a more balanced distribution.

[81]:
```
# Convert salary into the logarithmic scale
baseball['salary'] = np.log(baseball['salary'])
```

[82]:
```
# b.
# checking for missing values
print(f"There are {baseball.isna().sum().sum()} values in the dataset")
```

```
There are 0 values in the dataset
```

[83]:
```
#b
# Among all the predictors, how many of them are continuous, integer counts,
# and categorical, respectively?
```

```python
# Count the number of continuous predictors
num_continuous = sum(baseball.dtypes == float)

# Count the number of integer count predictors
num_integer_counts = sum(baseball.dtypes == int)

# Count the number of categorical predictors
num_categorical = sum(baseball.dtypes == object)

# Print the results
print(f"Number of continuous predictors: {num_continuous}")
print(f"Number of integer count predictors: {num_integer_counts}")
print(f"Number of categorical predictors: {num_categorical}")
```

```
Number of continuous predictors: 3
Number of integer count predictors: 14
Number of categorical predictors: 1
```

[84]:
```python
baseball.dtypes
```

[84]:
```
salary              float64
batting.avg         float64
OBP                 float64
runs                  int64
hits                  int64
doubles               int64
triples               int64
homeruns              int64
RBI                   int64
walks                 int64
strike.outs           int64
stolen.bases          int64
errors                int64
free.agency.elig      int64
free.agent.91         int64
arb.elig              int64
arb.91                int64
name                 object
dtype: object
```

It might be the case some of the categories were converted into numerical values using one-hot encoding. But we no longer have to worry about that.

2 Linear Regression with Variable Selection/Regularization

[85]:
```python
# Partition the data randomly into two sets: the training data D0 and the test
# data D1 with a ratio of about 2:1. Set random_state = 42.
from sklearn.model_selection import train_test_split

#D0, D1 = train_test_split(baseball, test_size=0.33, random_state=42)
# split the data
X = baseball.drop(['salary','name'],axis = 1)
y= baseball.salary
X_D0, X_D1, y_D0,y_D1 = train_test_split(X, y, test_size=0.33, random_state=42)
```

3 Ridge Regression

[86]:
```python
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
from sklearn.metrics import mean_squared_error
```

[87]:
```python
alphas = 10**np.linspace(10,-2,100)*0.5
alphas
```

[87]:
```
array([5.00000000e+09, 3.78231664e+09, 2.86118383e+09, 2.16438064e+09,
       1.63727458e+09, 1.23853818e+09, 9.36908711e+08, 7.08737081e+08,
       5.36133611e+08, 4.05565415e+08, 3.06795364e+08, 2.32079442e+08,
       1.75559587e+08, 1.32804389e+08, 1.00461650e+08, 7.59955541e+07,
       5.74878498e+07, 4.34874501e+07, 3.28966612e+07, 2.48851178e+07,
       1.88246790e+07, 1.42401793e+07, 1.07721735e+07, 8.14875417e+06,
       6.16423370e+06, 4.66301673e+06, 3.52740116e+06, 2.66834962e+06,
       2.01850863e+06, 1.52692775e+06, 1.15506485e+06, 8.73764200e+05,
       6.60970574e+05, 5.00000000e+05, 3.78231664e+05, 2.86118383e+05,
       2.16438064e+05, 1.63727458e+05, 1.23853818e+05, 9.36908711e+04,
       7.08737081e+04, 5.36133611e+04, 4.05565415e+04, 3.06795364e+04,
       2.32079442e+04, 1.75559587e+04, 1.32804389e+04, 1.00461650e+04,
       7.59955541e+03, 5.74878498e+03, 4.34874501e+03, 3.28966612e+03,
       2.48851178e+03, 1.88246790e+03, 1.42401793e+03, 1.07721735e+03,
       8.14875417e+02, 6.16423370e+02, 4.66301673e+02, 3.52740116e+02,
       2.66834962e+02, 2.01850863e+02, 1.52692775e+02, 1.15506485e+02,
       8.73764200e+01, 6.60970574e+01, 5.00000000e+01, 3.78231664e+01,
       2.86118383e+01, 2.16438064e+01, 1.63727458e+01, 1.23853818e+01,
       9.36908711e+00, 7.08737081e+00, 5.36133611e+00, 4.05565415e+00,
       3.06795364e+00, 2.32079442e+00, 1.75559587e+00, 1.32804389e+00,
       1.00461650e+00, 7.59955541e-01, 5.74878498e-01, 4.34874501e-01,
       3.28966612e-01, 2.48851178e-01, 1.88246790e-01, 1.42401793e-01,
       1.07721735e-01, 8.14875417e-02, 6.16423370e-02, 4.66301673e-02,
       3.52740116e-02, 2.66834962e-02, 2.01850863e-02, 1.52692775e-02,
```

```
1.15506485e-02, 8.73764200e-03, 6.60970574e-03, 5.00000000e-03])
```

[88]:
```python
ridge = Ridge(normalize = True)
coefs = []

for a in alphas:
    ridge.set_params(alpha = a)
    ridge.fit(X, y)
    coefs.append(ridge.coef_)

np.shape(coefs)
```

[88]:
```
(100, 16)
```

[89]:
```python
# Perform Ridge Regression with cross-validation
ridge_cv = RidgeCV(alphas=alphas, cv=10, scoring='neg_mean_squared_error',␣
 ↪normalize=True)
ridge_cv.fit(X_D0, y_D0)
ridge_cv_alpha = ridge_cv.alpha_
```

[90]:
```python
# Perform Lasso Regression with cross-validation
lasso_cv = LassoCV(alphas=alphas, cv=10, normalize=True)
lasso_cv.fit(X_D0, y_D0)
lasso_cv_alpha = lasso_cv.alpha_
```

[92]:
```python
# Perform elastic net Regression with cross-validation
from sklearn.linear_model import ElasticNetCV
elastic_net_cv = ElasticNetCV(alphas=alphas, cv=10, normalize=True)
elastic_net_cv.fit(X_D0, y_D0)
elastic_net_cv_alpha = elastic_net_cv.alpha_
```

[95]:
```python
# Choose the best model based on the lowest mean squared error
print("The best model is:")
if ridge_cv_alpha < lasso_cv_alpha and ridge_cv_alpha < elastic_net_cv_alpha:
    print("Ridge Regression")
elif lasso_cv_alpha < ridge_cv_alpha and lasso_cv_alpha < elastic_net_cv_alpha:
    print("Lasso Regression")
else:
    print("Elastic Net Regression")
```

```
The best model is:
Elastic Net Regression
```

We performed a 10-fold cross-validation to select the tuning parameter, lambda, for both Ridge Regression and Lasso. The cross-validation is done using different sets of lambdas specified in the alphas array. The goal is to find the lambda value that results in the most efficient prediction, as measured by the negative mean squared error. The RidgeCV, LassoCV, and ElasticNetCV functions are used for this purpose, with the cv parameter set to 10 for 10-fold cross-validation. The selected lambda values are then used to fit the Ridge, Lasso, and Elastic Net models, respectively.

[98]:
```python
# Output the necessary fitting results for each model
print("Ridge Regression Model:")
print(pd.Series(ridge_cv.coef_, index=X_D0.columns))
print(f"Slope parameter estimates: {ridge_cv.intercept_}")
```

```
Ridge Regression Model:
batting.avg        -0.026696
OBP                -1.193753
runs               -0.003033
hits                0.006111
doubles            -0.001098
triples            -0.010288
homeruns            0.003594
RBI                 0.009204
walks               0.005465
strike.outs        -0.003281
stolen.bases        0.001835
errors             -0.010493
free.agency.elig    1.612707
free.agent.91      -0.405020
arb.elig            1.283800
arb.91             -0.083832
dtype: float64
Slope parameter estimates: 5.301162845962
```

[99]:
```python
# lasso model
print("Lasso Model:")
print(pd.Series(lasso_cv.coef_, index=X_D0.columns))
print(f"Slope parameter estimates: {lasso_cv.intercept_}")
```

```
Lasso Model:
batting.avg         0.000000
OBP                 0.000000
runs                0.000000
hits                0.003842
doubles             0.000000
triples             0.000000
homeruns            0.000000
RBI                 0.008987
walks               0.000302
strike.outs         0.000000
stolen.bases        0.000000
errors             -0.000000
free.agency.elig    1.383049
free.agent.91      -0.074505
```

```
arb.elig             1.024780
arb.91               0.000000
dtype: float64
Slope parameter estimates: 5.019733377472774
```

[104]:
```
# elastic net
print("Elastic Net Model:")
print(pd.Series(elastic_net_cv.coef_, index=X_D0.columns))
print(f"Slope parameter estimates: {elastic_net_cv.intercept_}")
```

```
Elastic Net Model:
batting.avg          0.000000
OBP                  0.000000
runs                 0.002721
hits                 0.002822
doubles              0.007864
triples              0.003741
homeruns             0.007062
RBI                  0.004739
walks                0.002874
strike.outs          0.000000
stolen.bases         0.000000
errors              -0.000000
free.agency.elig     0.777097
free.agent.91        0.000000
arb.elig             0.524915
arb.91               0.016044
dtype: float64
Slope parameter estimates: 5.209651359072835
```

[106]:
```
# Apply the models to the test data D1
from sklearn.metrics import mean_squared_error
mean_squared_error(y_D1,ridge_cv.predict(X_D1))
```

[106]: 0.32470580502716334

[107]:
```
mean_squared_error(y_D1,lasso_cv.predict(X_D1))
```

[107]: 0.33112906110166923

[108]:
```
mean_squared_error(y_D1,elastic_net_cv.predict(X_D1))
```

[108]: 0.4217061183178383

The ridge has the least MSE, so this is the best model.

[109]:
```
fit_final = ridge_cv.fit(baseball.drop(['salary', 'name'], axis=1),␣
  ↪baseball['salary'])
```

[110]:
```
print(pd.Series(fit_final.coef_, index=baseball.drop(['salary', 'name'],␣
↪axis=1).columns))
print(f"Slope parameter estimates: {fit_final.intercept_}")
```

```
batting.avg          0.596579
OBP                 -1.779626
runs                 0.002238
hits                 0.004542
doubles              0.000343
triples             -0.015405
homeruns             0.005968
RBI                  0.008480
walks                0.004629
strike.outs         -0.004994
stolen.bases         0.004018
errors              -0.007587
free.agency.elig     1.506393
free.agent.91       -0.203964
arb.elig             1.245256
arb.91              -0.055978
dtype: float64
Slope parameter estimates: 5.360133580469188
```

3.1 Interpreting the coefficient estimates:

- The coefficient estimates represent the change in the predicted salary for a one-unit increase in each predictor variable, holding all other variables constant. For example, by improving the batting average by 100, the salary increase by 597.
- A positive coefficient indicates that an increase in the corresponding predictor variable is associated with an increase in the predicted salary, while a negative coefficient indicates the opposite.
- The magnitude of the coefficient represents the strength of the relationship between the predictor variable and the predicted salary. A larger magnitude indicates a stronger relationship.
- The intercept term represents the predicted salary when all predictor variables are zero.

[111]:
```
new_test = pd.read_csv('bb92-test-2.csv')
new_test.head()
```

[111]:

	batting.avg	OBP	runs	hits	doubles	triples	homeruns	RBI	walks
0	0.234	0.346	51	45	19	2	9	50	37
1	0.281	0.354	67	70	11	0	1	8	25
2	0.243	0.350	84	102	30	0	4	50	65
3	0.286	0.138	10	140	4	4	8	11	23
4	0.194	0.339	38	113	16	3	0	34	5

	strike.outs	stolen.bases	errors	free.agency.elig	free.agent.91
0	133	34	10	0	0
1	65	4	9	1	0

```
2          107            41       5                  1                0
3           48             6       0                  0                0
4           60             0       6                  0                0

   arb.elig  arb.91
0         0       0
1         0       0
2         0       0
3         0       0
4         0       0
```

[112]:
```
new_test_predictions = fit_final.predict(new_test)
new_test['logsalary'] = new_test_predictions
new_test.head()
```

[112]:
```
   batting.avg    OBP  runs  hits  doubles  triples  homeruns  RBI  walks  \
0        0.234  0.346    51    45       19        2         9   50     37
1        0.281  0.354    67    70       11        0         1    8     25
2        0.243  0.350    84   102       30        0         4   50     65
3        0.286  0.138    10   140        4        4         8   11     23
4        0.194  0.339    38   113       16        3         0   34      5

   strike.outs  stolen.bases  errors  free.agency.elig  free.agent.91  \
0          133            34      10                 0              0
1           65             4       9                 1              0
2          107            41       5                 1              0
3           48             6       0                 0              0
4           60             0       6                 0              0

   arb.elig  arb.91  logsalary
0         0       0   5.223746
1         0       0   6.688549
2         0       0   7.391396
3         0       0   5.915070
4         0       0   5.396459
```

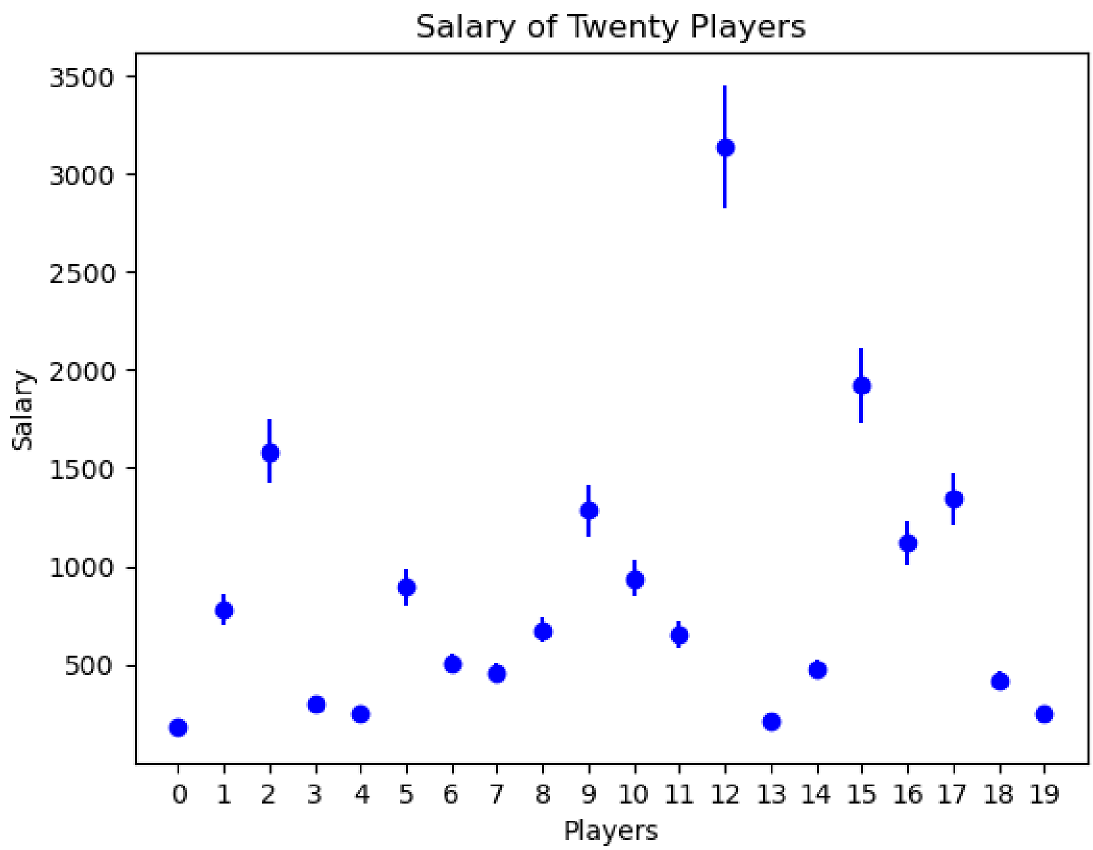
[113]:
```
new_test['salary'] = np.exp(new_test['logsalary'])
new_test = new_test[['salary'] + list(new_test.columns[:-1])]
new_test.head(5)
```

[113]:
```
        salary  batting.avg    OBP  runs  hits  doubles  triples  homeruns  \
0   185.628249        0.234  0.346    51    45       19        2         9
1   803.156422        0.281  0.354    67    70       11        0         1
2  1621.969166        0.243  0.350    84   102       30        0         4
3   370.580286        0.286  0.138    10   140        4        4         8
4   220.623827        0.194  0.339    38   113       16        3         0
```

```
   RBI  walks  strike.outs  stolen.bases  errors  free.agency.elig  \
0   50     37          133            34      10                 0
1    8     25           65             4       9                 1
2   50     65          107            41       5                 1
3   11     23           48             6       0                 0
4   34      5           60             0       6                 0

   free.agent.91  arb.elig  arb.91  logsalary
0              0         0       0   5.223746
1              0         0       0   6.688549
2              0         0       0   7.391396
3              0         0       0   5.915070
4              0         0       0   5.396459
```

[48]:

```python
import matplotlib.pyplot as plt

# Error bar plot
plt.errorbar(range(20), new_test['salary'], yerr=0.1*new_test['salary'],␣
 ↪fmt='o', color='b')
plt.xlabel('Players')
plt.ylabel('Salary')
plt.title('Salary of Twenty Players')
plt.xticks(range(20), new_test.index)
plt.show()
```


Salary of Twenty Players
Salary
3500
3000
2500
2000
1500
1000
500
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Players