cmps320_assignment3_SammanBhetwal

October 21, 2024

[3]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as smf
import warnings

warnings.filterwarnings('ignore')

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
from sklearn.preprocessing import MinMaxScaler
```

1 1.1 Exploratory data Analysis

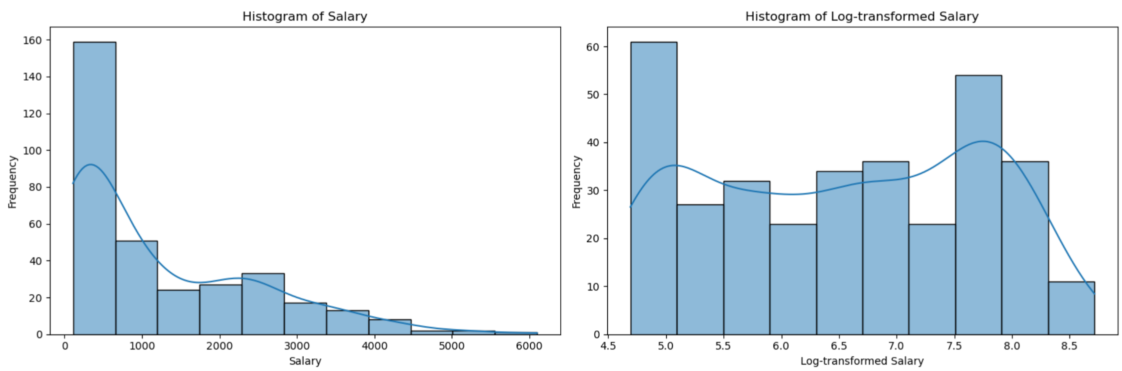
1.0.1 a) Obtain the histograms of both salary and the logarithm (natural base) of salary and comment. Proceed with the log-transformed salary from this step on.

[8]:
```python
baseball_data = 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_data.head()
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

# Plot the original salary distribution
sns.histplot(baseball_data['salary'], ax=ax1, kde=True)
ax1.set_title('Histogram of Salary')
ax1.set_xlabel('Salary')
ax1.set_ylabel('Frequency')
```

```
# Plot the log-transformed salary distribution
sns.histplot(np.log(baseball_data['salary']), ax=ax2, kde=True)
ax2.set_title('Histogram of Log-transformed Salary')
ax2.set_xlabel('Log-transformed Salary')
ax2.set_ylabel('Frequency')

plt.tight_layout()
plt.show()
```



The left graph(salary) is right-skewed which means there are few players that have high salary. On the other hand, the right graph, log-transformed, is distributed normally which indicates that the variance is stabilized. This helps in linear regression.

1.0.2 b) Inspect the data and answer these questions: Are there any missing data? Among all the predictors, how many of them are continuous, integer counts, and categorical, respectively?

[15]:
```
missing_values = baseball_data.isnull().sum()
missing_values
```

[15]:
```
salary          0
batting.avg     0
OBP             0
runs            0
hits            0
doubles         0
triples         0
homeruns        0
RBI             0
walks           0
strike.outs     0
stolen.bases    0
errors          0
```

```
free.agency.elig    0
free.agent.91       0
arb.elig            0
arb.91              0
name                0
dtype: int64
```

We can see form the data that there are no missing data

[17]:
```
# we are trying to see the data types now
baseball_data.dtypes
```

[17]:
```
salary                int64
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
```

The output above tells us that all the columns except "name" are either continuous or integer values. Regardless of the output, we will still analyze the date further to identify if some of the other columns are categorical.

[19]:
```
baseball_data.head()
```

[19]:
```
   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
```

After further analyzing the data we can see that name is a categorical data for sure. ON the other hand, free.agency.elig, free.agent.91, arb.elig and arb.91 only seem to contain the values of 0 and 1 for the first 10 entries. This leads me to stipulate that these might also be categorical. We can confirm this by counting each entries in these columns

[20]:
```
#printing the columns
print(baseball_data['free.agency.elig'].value_counts())
print(baseball_data['free.agent.91'].value_counts())
print(baseball_data['arb.elig'].value_counts())
print(baseball_data['arb.91'].value_counts())
```

```
0    203
1    134
Name: free.agency.elig, dtype: int64
0    298
1     39
Name: free.agent.91, dtype: int64
0    272
1     65
Name: arb.elig, dtype: int64
0    327
1     10
Name: arb.91, dtype: int64
```

The above output tells us that the last five columns are all categorical. batting.avg and OBP are continuous. Rest of the predictors are integer values.

1.2 Linear Regression with Variable Selection/Regularization:

1.2.1 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.

[58]:
```
X = baseball_data.drop(['name','salary'], axis = 1)
Y= baseball_data['salary']
scaler = MinMaxScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X))
```

[59]:
```
X_D0, X_D1, Y_D0,Y_D1 = train_test_split(X_scaled, Y, test_size=0.
↪33,random_state=42)
```

1.2.2 Using the training data D0, apply three variable selection/ regularization methods of your choice and identify your 'best' models accordingly.

[60]:
```
#### Ridge Regression
```

[61]:
```
alphas = 10**np.linspace(10,-2,100)*0.5
```

[62]:
```
ridge_cv = RidgeCV(alphas = alphas, cv = 10, scoring='neg_mean_squared_error')
ridge_cv.fit(X_D0,y_D0)
best_alpha = ridge_cv.alpha_
ridge = Ridge(alpha = best_alpha)
ridge.fit(X_D0, y_D0)
```

[62]:
```
Ridge(alpha=1.004616501282523)
```

[63]:
```
## Lasso
```

[64]:
```
lasso_cv = LassoCV(alphas = None, cv = 10, max_iter = 100000)
lasso_cv.fit(X_D0, y_D0)
lasso_cv.alpha_
lasso = Lasso(alpha=lasso_cv.alpha_)
lasso.fit(X_D0, y_D0)
```

[64]:
```
Lasso(alpha=3.5956521481481483)
```

[32]:
```
#Best Subset
```

[33]:
```
import itertools
import time
import statsmodels.api as sm
```

[65]:
```
def processTheSubset(feature_set):
    # Fit the model on feature_set and calculate RSS
    our_model = sm.OLS(y,X[list(feature_set)])
    regr = our_model.fit()
    RSS = ((regr.predict(X[list(feature_set)]) - y) ** 2).sum()
    return {"model":regr, "RSS":RSS}
```

[66]:
```
def getBest(k):
    tic = time.time()

    results = []

    for combo in itertools.combinations(X.columns, k):
        results.append(processTheSubset(combo))

    # we will now Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)
```

```
    # Now we will be Choosing the model with the highest RSS
    best_model = models.loc[models['RSS'].argmin()]

    toc = time.time()
    print("Processed", models.shape[0], "models on", k, "predictors␣
↪in",(toc-tic), "seconds.")
    # Return the best model, along with some other useful information about␣
↪the␣model
    return best_model
```

[68]:
```
models_best = pd.DataFrame(columns=["RSS", "model"])
tic = time.time()
for i in range(1,8):
    models_best.loc[i] = getBest(i)
toc = time.time()
print("Total elapsed time:", (toc-tic), "seconds.")
```

```
Processed 16 models on 1 predictors in 0.042380332946777344 seconds.
Processed 120 models on 2 predictors in 0.1830451488494873 seconds.
Processed 560 models on 3 predictors in 0.95591139793396 seconds.
Processed 1820 models on 4 predictors in 3.098555088043213 seconds.
Processed 4368 models on 5 predictors in 7.825338363647461 seconds.
Processed 8008 models on 6 predictors in 15.17093276977539 seconds.
Processed 11440 models on 7 predictors in 23.673619508743286 seconds.
Total elapsed time: 51.20165801048279 seconds.
```

[70]:
```
print(models_best) #printing
```

```
                RSS                                              model
1  285835104.427972  <statsmodels.regression.linear_model.Regressio…
2  214899595.709205  <statsmodels.regression.linear_model.Regressio…
3  191199900.970062  <statsmodels.regression.linear_model.Regressio…
4   177881297.85424  <statsmodels.regression.linear_model.Regressio…
5  169831836.397024  <statsmodels.regression.linear_model.Regressio…
6  162885295.270666  <statsmodels.regression.linear_model.Regressio…
7  160118089.576219  <statsmodels.regression.linear_model.Regressio…
```

[71]:
```
print(models_best.loc[2, "model"].summary())
```

```
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:                 salary   R-squared (uncentered):
0.794
Model:                            OLS   Adj. R-squared (uncentered):
0.793
Method:                 Least Squares   F-statistic:
644.6
Date:                Mon, 21 Oct 2024   Prob (F-statistic):
```

```
1.44e-115
Time:                        22:23:20   Log-Likelihood:
-2730.3
No. Observations:                 337   AIC:
5465.
Df Residuals:                     335   BIC:
5472.
Df Model:                           2
Covariance Type:            nonrobust
================================================================================
====
                       coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----
RBI                 20.7213      1.092     18.980      0.000      18.574
22.869
free.agency.elig   964.9391     91.762     10.516      0.000     784.437
1145.442
==============================================================================
Omnibus:                       30.723   Durbin-Watson:                   1.331
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               41.958
Skew:                           0.650   Prob(JB):                     7.74e-10
Kurtosis:                       4.139   Cond. No.                         111.
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[72]:

```python
print(getBest(16)["model"].summary())
```

```
Processed 1 models on 16 predictors in 0.007002115249633789 seconds.
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:                 salary   R-squared (uncentered):
0.852
Model:                            OLS   Adj. R-squared (uncentered):
0.844
Method:                 Least Squares   F-statistic:
115.3
Date:                Mon, 21 Oct 2024   Prob (F-statistic):
1.86e-122
Time:                        22:23:27   Log-Likelihood:
-2674.6
```

```
No. Observations:                 337   AIC:
5381.
Df Residuals:                     321   BIC:
5442.
Df Model:                          16
Covariance Type:            nonrobust
================================================================================
====
                        coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----
batting.avg        3116.6362   2708.000      1.151      0.251   -2211.033
8444.306
OBP               -2890.3326   2175.631     -1.329      0.185   -7170.630
1389.965
runs                  7.1570      5.638      1.269      0.205      -3.935
18.249
hits                 -2.8849      3.297     -0.875      0.382      -9.371
3.602
doubles               1.3814      8.604      0.161      0.873     -15.545
18.308
triples             -18.5295     21.609     -0.857      0.392     -61.043
23.984
homeruns             18.1385     12.411      1.461      0.145      -6.279
42.556
RBI                  17.6778      5.049      3.501      0.001       7.745
27.611
walks                 4.9268      4.321      1.140      0.255      -3.575
13.429
strike.outs          -8.9759      1.947     -4.609      0.000     -12.807
-5.145
stolen.bases         12.9717      4.709      2.755      0.006       3.707
22.236
errors               -9.2327      7.479     -1.235      0.218     -23.946
5.481
free.agency.elig  1383.0915    107.430     12.874      0.000    1171.735
1594.448
free.agent.91     -278.0038    137.459     -2.022      0.044    -548.438
-7.569
arb.elig            794.0206    117.162      6.777      0.000     563.518
1024.523
arb.91              345.6490    241.430      1.432      0.153    -129.335
820.633
==============================================================================
Omnibus:                       32.242   Durbin-Watson:                   1.560
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               65.504
Skew:                           0.524   Prob(JB):                     5.97e-15
```

```
Kurtosis:                       4.889   Cond. No.                     1.39e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[3] The condition number is large, 1.39e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[40]:
```python
models_best.loc[2, "model"].rsquared
```

[40]:
```
0.7937560345339563
```

[73]:
```python
models_best.apply(lambda row: row[1].rsquared, axis=1)
```

[73]:
```
1    0.725678
2    0.793756
3    0.816501
4    0.829283
5    0.837009
6    0.843675
7    0.846331
dtype: float64
```

1.2.3 Report each variable selection method's essential steps and key quantities. Ridge

1. Alpha determination: • Initially I set up a range of alpha values in logarithmic scale between $10\hat{}10 \times 0.5$ and $10\hat{}(-2) \times 0.5$. these values were provided in order to provide a spectrum of penalty strengths. Then by using RidgeCV, I performed 10-fold cross-validation to find the optimal alpha.This optimal alpha is based on minimizing the negative mean squared error. • After fitting RidgeCV to the scaled data X_D0, the best alpha value was retrieved using ridge_cv.alpha
2. Fitting the Optimal Model: • After initializing a Ridge regression model with the best alpha value obtained from the cross-validation, the model was then trained on the scaled data X_D0.

Key Quantities: best_alpha best_alpha represents the optimal alpha value determined by the cross-validation. -ridge: The ridge eregression model trained using the optimal aplha

Lasso

1.0.3 1. Alpha Determination

• I used LassoCV to perform a 10-fold cross-validation. this helped me determine the best alpha. • To ensure convergence, I set the max_iter parameter to 100,000 . • After fitting LassoCV to the scaled data X_D0, the optimal alpha value was extracted using lassocv.alpha_.

2.

Fitting the Optimal Model: • I initialized Lasso regression model with the optimal alpha value obtained from the previous step. • This model was then trained on the scaled data X_D0.

Key Quantities:

- lassocv.alpha_: Gives the optimal alpha value determined by the crossvalidation. - lasso: The Lasso regression model fitted using the optimal alpha.

Best Subset Selection

1. Process Subset: • when you have a given feature set, this fits a linear regression model. • Calculate and return its RSS.
2. Find Best Model for k Features: • Evaluates all the combinations of k predictors and Identifies the model with the smallest RSS.
3. Iterate Over Predictors: • Stores the best model's RSS for predictors ranging from 1 to 7 in models_best.
4. Output: • Display the models_best DataFrame, Shows the summary for the best model with 2 predictors, and extracts R-squared values for all best models.

Key Quantities: - RSS: Measure of model fit. - models_best: Best models for each predictor count.

1.2.4 Output the necessary fitting results for each model, e.g., selected variables and their corresponding slope parameter estimates.

Ridge

[47]:
```
pd.Series(ridge.coef_,index=X.columns)
```

[47]:
```
batting.avg            79.868560
OBP                   -91.174534
runs                  127.726059
hits                  371.547807
doubles               -70.270766
triples               -43.116870
homeruns              871.018638
RBI                  1523.035924
walks                 207.474137
strike.outs          -748.973837
stolen.bases          620.365473
errors               -407.798299
free.agency.elig     1526.027613
free.agent.91        -574.295685
arb.elig              857.493975
arb.91                107.060700
dtype: float64
```

lasso

[48]:
```
pd.Series(lasso.coef_,index=X.columns)
```

[48]:
```
batting.avg             0.000000
OBP                     0.000000
runs                    0.000000
hits                    0.000000
doubles                -0.000000
triples                -0.000000
homeruns              384.411628
RBI                  2381.199547
walks                   0.000000
strike.outs          -669.258725
stolen.bases          658.934382
errors               -332.688293
free.agency.elig     1569.862872
free.agent.91        -607.241458
arb.elig              870.609709
arb.91                  0.000000
dtype: float64
```

Best Subset

[49]:
```
print(getBest(16)["model"].summary())
```

```
Processed 1 models on 16 predictors in 0.006260395050048828 seconds.
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:                 salary   R-squared (uncentered):
0.852
Model:                            OLS   Adj. R-squared (uncentered):
0.844
Method:                 Least Squares   F-statistic:
115.3
Date:                Mon, 21 Oct 2024   Prob (F-statistic):
1.86e-122
Time:                        22:02:44   Log-Likelihood:
-2674.6
No. Observations:                 337   AIC:
5381.
Df Residuals:                     321   BIC:
5442.
Df Model:                          16
Covariance Type:            nonrobust
================================================================================
====
                       coef    std err          t      P>|t|      [0.025
0.975]
```

```
------------------------------------------------------------------------------------
----
batting.avg       3116.6362   2708.000      1.151      0.251   -2211.033
8444.306
OBP              -2890.3326   2175.631     -1.329      0.185   -7170.630
1389.965
runs                 7.1570      5.638      1.269      0.205      -3.935
18.249
hits                -2.8849      3.297     -0.875      0.382      -9.371
3.602
doubles              1.3814      8.604      0.161      0.873     -15.545
18.308
triples            -18.5295     21.609     -0.857      0.392     -61.043
23.984
homeruns            18.1385     12.411      1.461      0.145      -6.279
42.556
RBI                 17.6778      5.049      3.501      0.001       7.745
27.611
walks                4.9268      4.321      1.140      0.255      -3.575
13.429
strike.outs         -8.9759      1.947     -4.609      0.000     -12.807
-5.145
stolen.bases        12.9717      4.709      2.755      0.006       3.707
22.236
errors              -9.2327      7.479     -1.235      0.218     -23.946
5.481
free.agency.elig  1383.0915    107.430     12.874      0.000    1171.735
1594.448
free.agent.91     -278.0038    137.459     -2.022      0.044    -548.438
-7.569
arb.elig           794.0206    117.162      6.777      0.000     563.518
1024.523
arb.91             345.6490    241.430      1.432      0.153    -129.335
820.633
==============================================================================
Omnibus:                       32.242   Durbin-Watson:                   1.560
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               65.504
Skew:                           0.524   Prob(JB):                     5.97e-15
Kurtosis:                       4.889   Cond. No.                     1.39e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[3] The condition number is large, 1.39e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

1.2.5 Apply the models to the test data D1. Output the mean squared error (MSE). Let's consider the one yielding the minimum MSE as the "best" final model.

[50]:
```
mean_squared_error(y_D1,ridge.predict(X_D1))
```

[50]: 613286.5676135737

[51]:
```
mean_squared_error(y_D1,lasso.predict(X_D1))
```

[51]: 634851.2731158076

1.2.6 Refit your "best" final model using the entire data, i.e., D0 ffi D1, Call it fit_final. Provide and interpret your final model's output (i.e., coefficient estimates).

[75]:
```
ridge_cv = RidgeCV(alphas = alphas,cv=10,scoring='neg_mean_squared_error')
ridge_cv.fit(X_scaled,y)
ridge_cv.alpha_
best_fit = Ridge(alpha = ridgecv.alpha_)
best_fit.fit(X_scaled,y)
pd.Series(best_fit.coef_,index = X.columns)
```

[75]:
```
batting.avg          163.113828
OBP                 -303.350662
runs                 710.632637
hits                 179.960800
doubles              161.372611
triples             -247.875466
homeruns            1004.619653
RBI                 1499.982178
walks                346.107463
strike.outs        -1218.984681
stolen.bases         731.714919
errors              -341.993032
free.agency.elig    1336.887941
free.agent.91       -252.183344
arb.elig             771.342459
arb.91               301.671693
dtype: float64
```

1.2.7 Model Deployment: Apply your final model to predict the logsalary for the new data set in the bb92-test.csv, which contains the performance data only for 20 players. Next, take the exponential of the predicted values to transform them back to regular salary values for better interpretation.

[76]:
```
new_csv_data = pd.read_csv('bb92-test-2.csv')
log_salary_pred = np.log(best_fit.predict(new_csv_data))
log_salary_pred
```

[76]:
```
array([ 7.9962895 ,  8.60848857, 11.33236277,  9.23811852, 10.18834499,
        9.46383059, 11.412498  , 12.34206445, 11.67664363, 11.06967922,
```

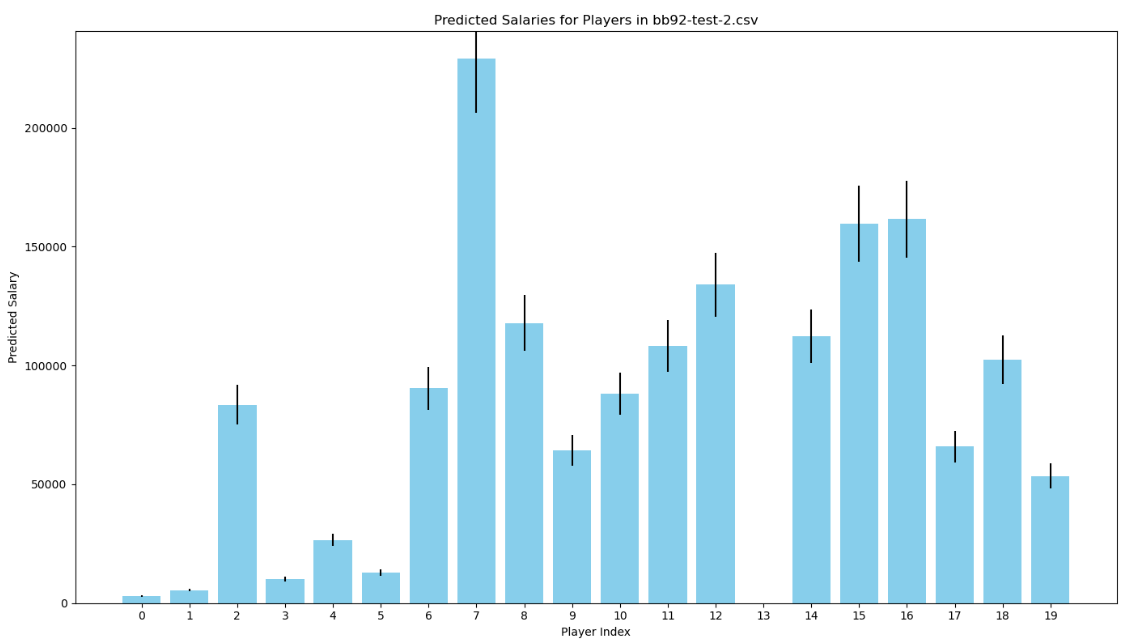
```
11.38676288, 11.59147501, 11.80596785,         nan, 11.63019568,
11.98088382, 11.99330216, 11.09469954, 11.53551462, 10.88666101])
```

[77]:
```python
salary_prediction = np.exp(log_salary_pred)
salary_prediction
```

[77]:
```
array([  2969.91764436,   5477.96287051,  83480.03446663,  10281.6756188 ,
        26591.44937781,  12885.14776413,  90445.07178148, 229134.50003961,
       117788.23049396,  64194.91106433,  88147.15237063, 108171.69508857,
       134049.9605264 ,            nan, 112442.32237549, 159673.08981629,
       161668.32831426,  65821.35019469, 102284.62217598,  53458.50445064])
```

[78]:
```python
# Using index numbers as player identifiers
player_indexes = range(len(new_data))

plt.figure(figsize=(14, 8))
plt.bar(player_indexes, salary_pred, color='skyblue', yerr=0.1 * salary_pred)
# Assuming a 10% error for demonstration
plt.xlabel('Player Index')
plt.ylabel('Predicted Salary')
plt.title('Predicted Salaries for Players in bb92-test-2.csv')
plt.xticks(ticks=player_indices)
plt.tight_layout()
plt.show()
```



The highest predicted salary among all belongs to the Player indexed at 7. Most of the players have predicted salaries in a similar range, with a few outliers (like the players indexed at 7, 15, and

16). These players have notably higher predicted salaries.

```
[ ]:
```