Comparisons of linear and classification methods on analysis and prediction of question earnings on the website Quora

1st, February 2022

I. Introduction

This report will analyze and predict the income from the Quora Partner Program: how much money can be earned from each posted question on the website Quora.

- Quora is a social question-and-answer website based in California, United States. To increase the advertisements revenues, Quora has devised a strategy called Partner Program in 2018, where users in this program are paid for regularly posting new questions. This helps increase the website's traffic volume.
- My Partner Program was in the Japanese language, so this report paper will only investigate the income scheme of the program in Japanese, which is probably inapplicable to programs in other languages.
- The structure of this report has 6 parts. The "Problem Formulation" section describes the purpose of this report, the datapoints, features, and label. The "Methods" section describes the dataset details, feature selection, the ML models, and their loss functions. The "Results" section shows the models' results and the comparison between their performance. The "Conclusions" section finalizes which model performs better and how questions should be formulated. The "Code Appendix" section attaches the Python code.

II. Problem Formulation

The problem in question is: given certain characteristics of the posted questions, how much income the question is likely to earn?

- To answer this question, this paper aims to systematically gather the necessary data and make a general analysis and prediction towards the income scheme of the Quora Partner Program.
- For each question datapoint, there are four features because they are easy to be collected in a highly automated manner, which is illustrated in the picture below

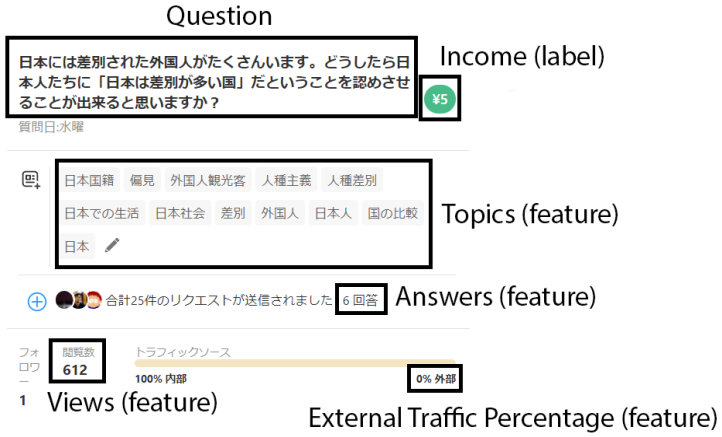


Figure 1: A posted question (datapoint) on the Quora website with many different features

Summary of datapoints, features, labels, and their data type

Datapoint: A single question with five features and one label

Features: There are 4 features: answers (integer), views (integer), external traffic percentage (integer), and topics of the question (string)

Label: Income of the question (integer - currency in Japanese yen). The income information is not available when the question is posted. It is only known one month after the question posting

III. Methods

3.1 Dataset

- **Source of the data:** The dataset is not available publicly because the income data is strictly exclusive to me. Nevertheless, the questions can be found at the URL provided below. Although there are over 7000 questions I have posted, I only choose the top 500 earning questions to study their properties and patterns. There is no missing data in any fields. After receiving the data file Quora has sent me, I proceed to filter the data so that it is readable by Python libraries, as well as manually input the features.

Link to the account on Quora: https://jp.quora.com/profile/Nguyen-Thanh-Luan

3.2 Feature Selection

I plot the data visualizations between the earnings label and the individual features. From the scatterplots in the annexed code, there seems to be a linear relationship between the views feature and earnings label, while there are no clear relations between the earnings and external traffic percentage. The answer feature seems to be weakly correlated with the label, but for the purpose of detailed analysis, it will also be chosen. The topics also correlate with earnings, because some certain topics are likely to earn more than the others.

=> Selected features: The answers, views, and topics features are chosen for this ML problem

3.3 Model (hypothesis space) selection

Due to the nature of the data points, I will separate the model selections into two types: the linear methods and the classification methods.

- The linear map will be used for modeling the features number of views and number of answers since in the data plots, they appear to be linearly related to the income. The two applied linear methods are Huber Regression and the Ridge Regression. Their hypothesis space will therefore be linear maps.
- The classification method will be used for modeling the question topics. For question topics, it contains 10 discrete topics and thus it is sensible to use classification methods for this feature. The two applied classification methods are logistic regression and support vector machine (SVM). These classification methods by definition use linear maps for their hypothesis space.

3.4 Loss functions selection

- For the ML models, their respective loss functions are chosen as follows:

- Huber Regression: The Huber loss functions (sklearn.linear_model.HuberRegressor)

There are a few extreme outliers in the dataset that will severely affect the result of the regression (Some questions have abnormally large earnings). This loss function is chosen because the Huber loss function is robust against outliers and thus provides a better prediction than a normal Linear Regression. [1]

- Ridge Regression: MSE loss regularized with L2-norm (sklearn.linear_model.RidgeRegression)

The label appears to have considerably large variance and the Linear Regression is expected to perform badly on validation sets. The Ridge Regression trades off an increase in bias with a decrease in variance so that the predictor would yield better results by introducing a penalty term [2]

- Logistic Regression: Logistic loss (sklearn.linear_model.LogisticRegression)

Logistic loss indicates how close the prediction probability is to the corresponding actual label. Particularly, the logistic loss is sensitive to outliers and there are many outliers in my dataset, which makes this loss function a good testing one for my model. [3]

- Support vector machine: Hinge loss (sklearn.svm.SVC)

There are many classes (10 topics) and therefore the boundaries are unclear. The hinge loss incorporates a margin from the boundary into the cost calculation. Thus, it will penalize the small margin if the margin from the decision boundary is not large enough, even if the observations are classified correctly [3]

3.5 Training and Validation Set

There are 500 datapoints in the dataset. First I split it into two sets: train-validation and testing sets of ratio 90% - 10%. Subsequently, I would split the train-validation set into training and validating sets using K-fold cross-validation with K = 5 as it is the standard commonly used fold number. Cross-validation usually delivers better results than a simple train-test split because it prevents overfitting. Moreover, on small datasets, the extra computational costs of cross-validation are not significant. [4]

IV. Results

4.1 Linear Methods

For the linear methods, the mean squared error in sklearn.metrics.mean_squared_error will be used to calculate the training and validation errors. Mean squared error (MSE) of a regression method measures the average squared differences between the actual and estimated values [5]. The best-chosen model minimizes the average validation error the most in the cross-validation. The obtained errors are:

	Training MSE Error		Validation MSE Error	
Features	Huber Regression	Ridge Regression	Huber Regression	Ridge Regression
Views	188288	181006	109014	102863
Answers	420880	326376	161739	198376

4.2 Comparison and selection of linear methods

From the MSE error table above, the validation error of the Ridge Regression is smaller for the views feature. On the other hand, the Huber Regression validation error is smaller for the feature Answer.

=> The chosen linear method is the Ridge Regression for feature Views and Huber Regression for feature Answers. These are the slope and intercept of their respective chosen model.

	Views	Answers		Views	Answers
Slope	0.016	9.249	Intercept	131.123	149.271

4.3 Classification Methods

For the classification methods, the accuracy score in sklearn.metrics.accuracy_score will be used to calculate the accuracy of the classifier models. Accuracy score computes the ratio of the accurately predicted features [6] Since there are 10 different groups and the accuracy of the model would be low, I merge the topics into three groups: Group 0 (News/Culture/Politics/Business/Career), Group 1 (Health/Psychology/Life/Relationship) and Group 2 (Entertainment/Technology/Education/ Fashion)

Similarly, the chosen accuracy is from the classifier model that maximizes the average validation accuracy in the cross-validation. The obtained accuracies are:

	Average Training Accuracy		Average Validation Accuracy	
Features	Logistic Regression	Support Vector Machine	Logistic Regression	Support Vector Machine
Topics	0.388	0.401	0.339	0.342

4.4 Comparison and selection of classification methods

From the accuracy score table above, both the training and validation accuracy of Support Vector Machine are greater than those of the Logistic Regression.

=> The chosen classifier method is the Support Vector Machine for feature Topics

4.5 Testing Set, testing errors, and accuracy

As stated above, the size of the testing set comprises 10% of the 500 datapoints or 50 datapoints, which has not been used in the cross-validation. The testing set helps provide an unbiased performance of the final fit model on the training set. The testing errors and accuracy have been finalized in the table below.

Features	Huber Regressor Error	Ridge Regressor Error		Logistic Regressor Accuracy	SVM Accuracy
Views	1218286	1218836	Topics	0.4	0.4
Answers	2305027	1705803			

V. Conclusion

5.1 Summary, result interpretations, and question formulation predictions

- Ridge regression on Views - Earnings: according to the Ridge Regression model, for every 1000 views a question receives, it earns an additional income of roughly 16¥. Since a question must gain user traffic to earn money, the intercept 131¥ of the regression line has little meaning because the number of views is 0.
- Huber regression on Answers - Earnings: according to the Huber Regression model, for every 1 answer a question receives, it earns an additional income of roughly 9.3¥. A question can earn money without answers on Quora and thus, for the top-earning questions with no answers, they earn 149.2¥ on average
- Support Vector Machine on Topics - Earnings: the predictions determined by SVM are given as:

Earnings: [0 20…160 180 200 220…1820 1840 1860 1880…1940 1960…2280 2300 2320 2340…10000]

Topic : [1 1 … 1 1 0 0 … 0 0 1 1 … 1 2 … 2 2 0 0 … 0]

By the categorization, it appears that group 0 and 1 topics dominate the lower part of income. On the other end, group 0 topics have the highest chance of reaching a high income. For group 2 questions, they may earn as much as group 1. Besides, group 2 is likely to earn a sufficiently large income like group 0.

- Question formulation prediction: Since views and answers are features that cannot be controlled, I can only depend on the question topics. It appears that Japanese people are usually interested in their own culture, daily life tips, recommendations on appearance, relationships, and work. A good strategy could be composing questions about hotly debated social problems, latest news, and product recommendations.

5.2 Results optimality and flaws

- Optimality: since the training and the validations errors of the Ridge Regression model are close for the views feature, the model does not overfit the training data, which can be attributed to the K-fold cross-validation. For SVM classifier, it performs moderately well at around one correct prediction out of three.
- Flaws: There are great discrepancies between the training and validation errors of the Huber Regression model for answers feature, suggesting that the model overfits the training data. For testing results, the testing MSE errors of both linear models are remarkably greater than their training and validation errors, which stems from the extreme outliers included in the testing set and thus results in poor performance.

5.3 Future directions and improvements on the ML methods

- First, more datapoints should be collected for the dataset. Because the chosen datapoints only represent the top 500 earning questions, they may not truly represent the whole trend of the 7000 posted questions.
- Secondly, there could be a better prediction if linear regression is a combination of various features. In that case, Lasso Regression can help to determine which features are important in explaining the earnings.
- Thirdly, there could probably not exist any linear relationships between the income and the view and answer features. In other words, the income is purely random. A more elaborate ML model will be needed to discover such a relationship if it truly exists.

VI. References

[1] Calculate Huber Loss using TensorFlow 2, available at
https://lindevs.com/calculate-huber-loss-using-tensorflow-2/
[2] Ridge Regression and Lasso Regression, available at
https://discuss.boardinfinity.com/t/ridge-regression-and-lasso-regression/6208
[3] Understanding Hinge Loss and the SVM Cost Function. Available at
https://programmathically.com/understanding-hinge-loss-and-the-svm-cost-function/
[4] Cross Validation. Available at:
https://www.kaggle.com/code/dansbecker/cross-validation/notebook
[5] How to Calculate Mean Squared Error (MSE) in Python. Available at:
https://vedexcel.com/how-to-calculate-mean-squared-error-mse-in-python/
[6] Accuracy Score. Available at
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

VII. Code appendix

Machine Learning Project

March 27, 2022

[1]:
```
# %config Completer.use_jedi = False  # enable code auto-completion
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import HuberRegressor, Ridge, LogisticRegression
from sklearn.model_selection import train_test_split, KFold
from sklearn.svm import SVC
from sklearn.metrics import mean_squared_error, accuracy_score, r2_score    #␣
↪function to calculate mean squared error
```

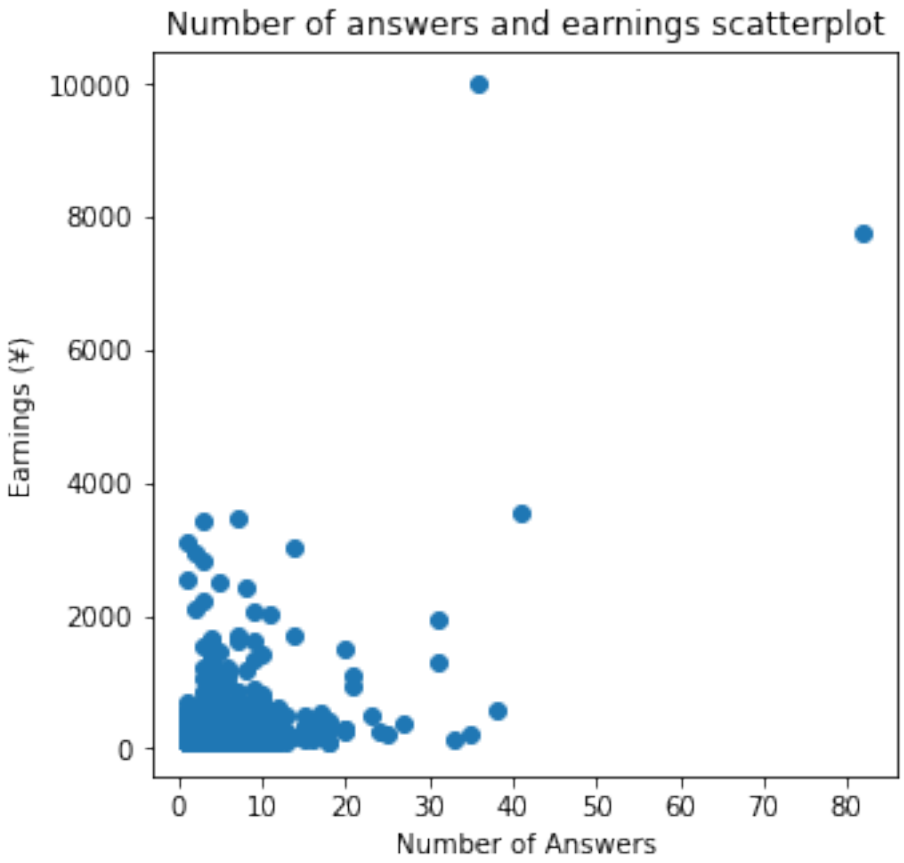
[2]:
```
df = pd.read_excel("earnings.xlsx")
print(df)
```

```
     Answers   Views                    Topic  External Traffic Percentage  \
0         36  247835            Social / News                            3
1         82  332393      Health / Psychology                            2
2         41  455160            Social / News                           19
3          7  106588       Culture / Politics                            2
4          3   65732            Social / News                            1
..       …     …                        …                          …
495        1    1134       Culture / Politics                           81
496        3    1339      Education / Science                           93
497        5    3159  Entertainment / Hobbies                           22
498       11    4139      Love / Relationship                            4
499        6    1068             Life / Habit                           94

     Earnings                                   Question details
0       10005  What's wrong with customs? You can play with y…
1        7785  It's very unpleasant to see someone eating out…
2        3542  I'm wondering if I should get the corona vacci…
3        3462  Why didn't the United States make Japan a poor…
4        3445  Do you think NASA has top secret files that ca…
..        …                                                 …
495        83  What do the Chinese think about the Tank Man i…
496        83  What happens when citric acid and carbonated w…
497        82              What are your favorite top 10 movies?
498        82                 Can you buy your heart with money?
499        82  What is the temperature of the air conditioner…
```

```
[500 rows x 6 columns]
```

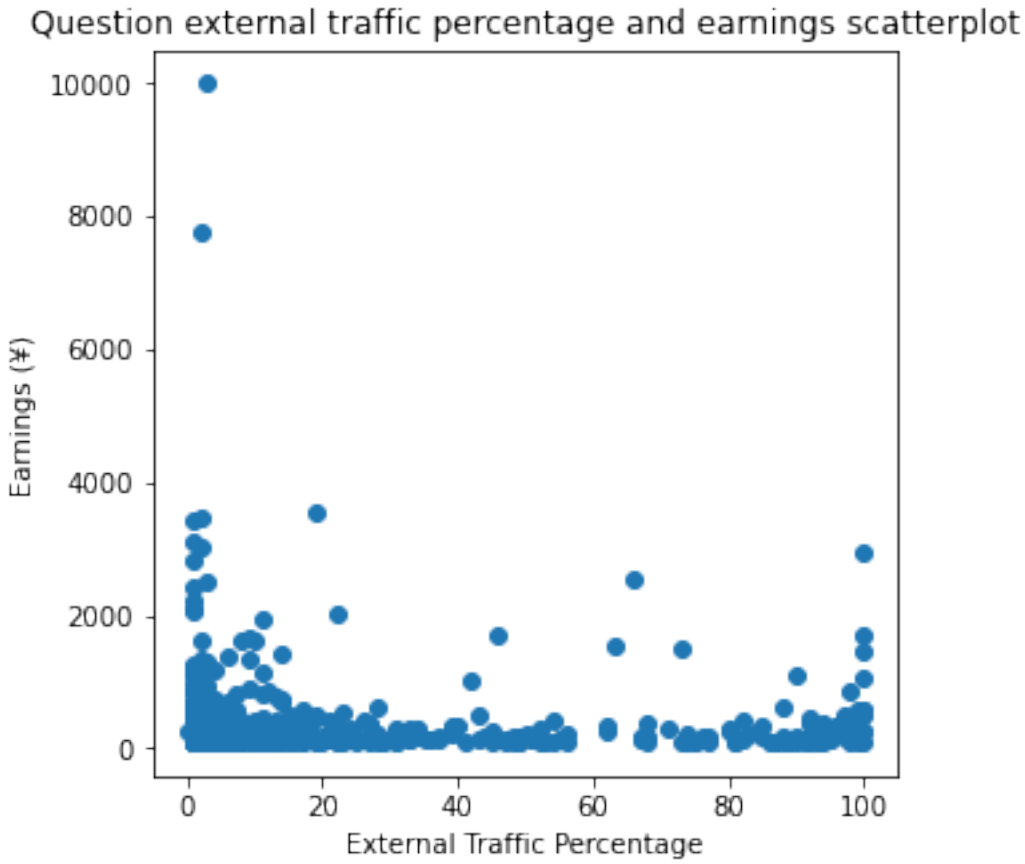
[3]:
```
external_traffic_percentage = df["External Traffic Percentage"].to_numpy()
topics = df["Topic"].to_numpy()
answers = df["Answers"].to_numpy()
views = df["Views"].to_numpy()
earnings = df["Earnings"].to_numpy()
```

[4]:
```
fig, axes = plt.subplots(figsize=(5,5))
axes.scatter(answers, earnings);
axes.set_xlabel("Number of Answers")
axes.set_ylabel("Earnings (¥)")
axes.set_title("Number of answers and earnings scatterplot")
plt.show()
```

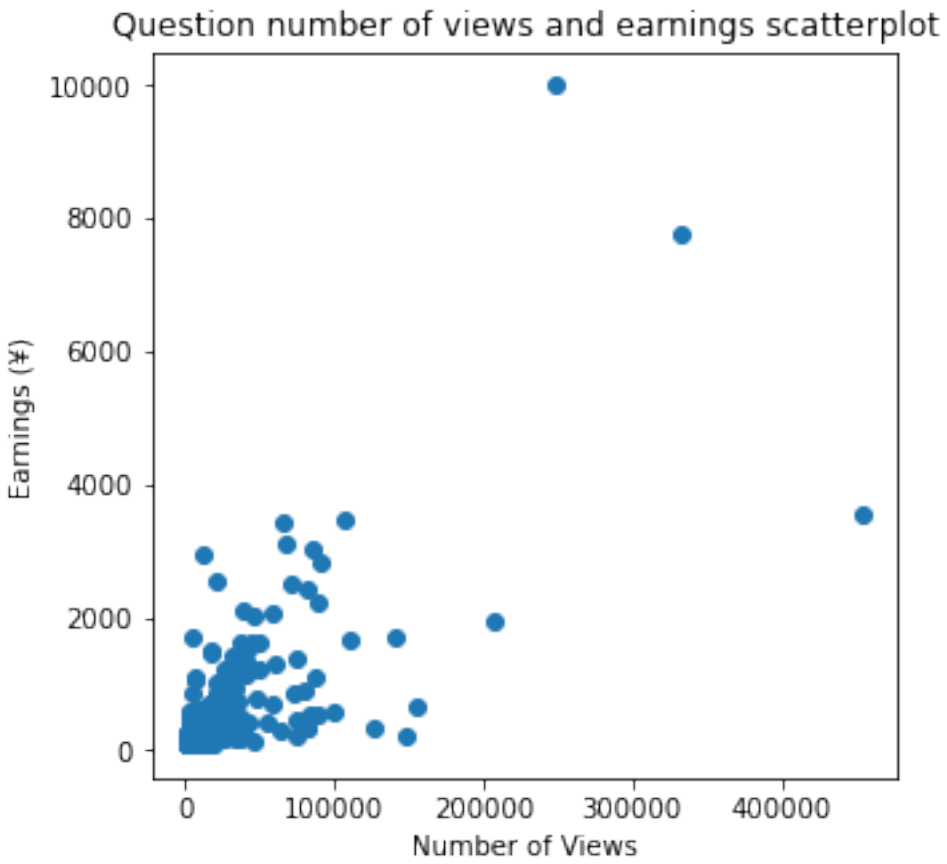


[5]:
```
fig, axes = plt.subplots(figsize=(5,5))
axes.scatter(external_traffic_percentage, earnings);
axes.set_xlabel("External Traffic Percentage")
axes.set_ylabel("Earnings (¥)")
```

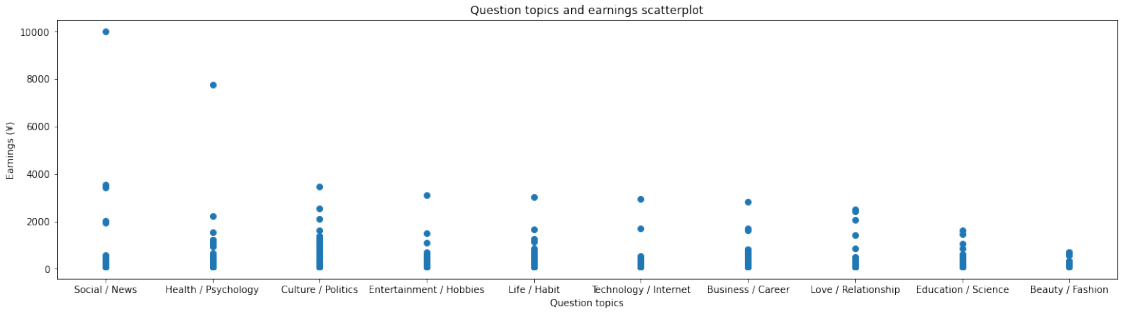
```
axes.set_title("Question external traffic percentage and earnings scatterplot")
plt.show()
```



[6]:
```
fig, axes = plt.subplots(figsize=(5,5))
axes.scatter(views, earnings);
axes.set_xlabel("Number of Views")
axes.set_ylabel("Earnings (¥)")
axes.set_title("Question number of views and earnings scatterplot")
plt.show()
```



[7]:
```
fig, axes = plt.subplots(1, 1, figsize=(20,5))
axes.scatter(topics, earnings);
axes.set_xlabel("Question topics")
axes.set_ylabel("Earnings (¥)")
axes.set_title("Question topics and earnings scatterplot")
plt.show()
```



[8]:
```
# Huber Regression, Ridge Regression model comparison function

def model_comparison(X, y, feature):
    tr_error = { "Huber": [], "Ridge": [] }
    val_error = { "Huber": [], "Ridge": [] }
    slope = { "Huber": [], "Ridge": [] }
    intercept = { "Huber": [], "Ridge": [] }
    predictors = { "Huber": [], "Ridge": [] }

    # In the first step we will split the data in testing and remaining dataset
    X_rem, X_test, y_rem, y_test = train_test_split(X, y, train_size=0.90,␣
 ↪random_state = 42)

    fig, axes = plt.subplots(5, 1, figsize=(5,25))

    K = 5  # Number of splits
    kf = KFold(n_splits=K, shuffle=True, random_state=42)    # Create a KFold␣
 ↪object with 'K' splits
    i = 0
    for train_indices, val_indices in kf.split(X_rem):
        X_train = X_rem[train_indices,:]    # Get the training set
        X_val = X_rem[val_indices,:]    # Get the validation set
        y_train = y_rem[train_indices]    # Get the training set
        y_val = y_rem[val_indices]    # Get the validation set

        # for Huber Regression model training
        HuberModel = HuberRegressor(epsilon = 1)
        fitHuberModel = HuberModel.fit(X_train, y_train)
        predictors["Huber"].append(fitHuberModel)
        slope["Huber"].append(fitHuberModel.coef_[0])
        intercept["Huber"].append(fitHuberModel.intercept_)
        y_pred_train_Huber = fitHuberModel.predict(X_train)
        # for Huber Regression validation
        y_pred_val_Huber = fitHuberModel.predict(X_val)
        tr_error["Huber"].append(mean_squared_error(y_train,␣
 ↪y_pred_train_Huber))
        val_error["Huber"].append(mean_squared_error(y_val, y_pred_val_Huber))

        # for Ridge Regression model training
        RidgeModel = Ridge(alpha=1, solver='svd')
        fitRidgeModel = RidgeModel.fit(X_train, y_train)
        predictors["Ridge"].append(fitRidgeModel)
        slope["Ridge"].append(fitRidgeModel.coef_[0])
        intercept["Ridge"].append(fitRidgeModel.intercept_)
        y_pred_train_Ridge = fitRidgeModel.predict(X_train)
        # for Ridge Regression validation
        y_pred_val_Ridge = fitRidgeModel.predict(X_val)
```

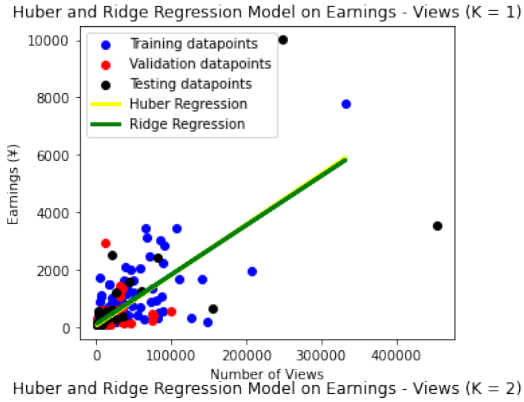
```
        tr_error["Ridge"].append(mean_squared_error(y_train,␣
↪y_pred_train_Ridge))
        val_error["Ridge"].append(mean_squared_error(y_val, y_pred_val_Ridge))

        # Start plotting
        axes[i].set_title("Huber and Ridge Regression Model on Earnings - " +␣
↪feature + " (K = " + str(i+1) + ")")
        axes[i].set_xlabel("Number of " + feature)
        axes[i].set_ylabel("Earnings (¥)")
        axes[i].scatter(X_train, y_train, color="blue", label = "Training␣
↪datapoints")
        axes[i].scatter(X_val, y_val, color="red", label = "Validation␣
↪datapoints")
        axes[i].scatter(X_test, y_test, color="black", label = "Testing␣
↪datapoints")
        axes[i].plot(X_train, y_pred_train_Huber, color="yellow", linewidth=3,␣
↪label = "Huber Regression")
        axes[i].plot(X_train, y_pred_train_Ridge, color="green", linewidth=3,␣
↪label = "Ridge Regression")
        axes[i].legend()
        i += 1

    # Printing out the result
    best_predictor_index_Huber = val_error["Huber"].
↪index(min(val_error["Huber"]))
    best_predictor_index_Ridge = val_error["Ridge"].
↪index(min(val_error["Ridge"]))
    y_pred_test_Huber = predictors["Huber"][best_predictor_index_Huber].
↪predict(X_test)
    y_pred_test_Ridge = predictors["Ridge"][best_predictor_index_Ridge].
↪predict(X_test)
    err_train = sum(tr_error["Huber"])/len(tr_error["Huber"])
    err_val = sum(val_error["Huber"])/len(val_error["Huber"])
    err_test = mean_squared_error(y_test, y_pred_test_Huber)

    print("Huber Regression. Error measured by MSE")
    print(f'Slopes for each K: {slope["Huber"]}')
    print(f'Slope by the best model predictor:␣
↪{slope["Huber"][best_predictor_index_Huber]}')
    print(f'Intercepts for each K: {intercept["Huber"]}')
    print(f'Intercept by the best model predictor:␣
↪{intercept["Huber"][best_predictor_index_Huber]}')
    print(f'Training error for each K: {tr_error["Huber"]}')
    print(f'Average training error: {err_train}')
    print(f'Training error by the best model predictor:␣
↪{tr_error["Huber"][best_predictor_index_Huber]}')
```

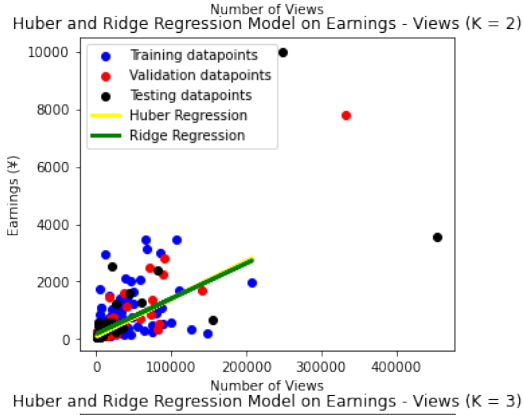
```
    print(f'Validation error for each K: {val_error["Huber"]}')
    print(f'Average validation error: {err_val}')
    print(f'Validation error by the best model predictor:␣
↪{val_error["Huber"][best_predictor_index_Huber]}')
    print(f'Testing error by the best model predictor: {err_test}\n')

    err_train = sum(tr_error["Ridge"])/len(tr_error["Ridge"])
    err_val = sum(val_error["Ridge"])/len(val_error["Ridge"])
    err_test = mean_squared_error(y_test, y_pred_test_Ridge)
    print("Ridge Regression. Error measured by MSE")
    print(f'Slopes for each K: {slope["Ridge"]}')
    print(f'Slope by the best model predictor:␣
↪{slope["Ridge"][best_predictor_index_Ridge]}')
    print(f'Intercepts for each K: {intercept["Ridge"]}')
    print(f'Intercept by the best model predictor:␣
↪{intercept["Ridge"][best_predictor_index_Ridge]}')
    print(f'Training error for each K: {tr_error["Ridge"]}')
    print(f'Average validation error: {err_val}')
    print(f'Training error by the best model predictor:␣
↪{tr_error["Ridge"][best_predictor_index_Ridge]}')
    print(f'Validation error for each K: {val_error["Ridge"]}')
    print(f'Average training error: {err_train}')
    print(f'Validation error by the best model predictor:␣
↪{val_error["Ridge"][best_predictor_index_Ridge]}')
    print(f'Testing error by the best model predictor: {err_test}\n')

    plt.show()
```

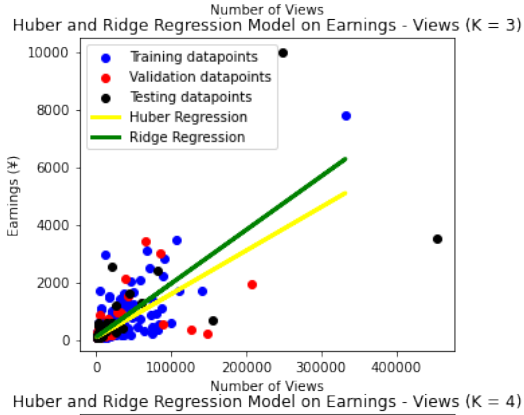
[9]:
```
# Comparing Huber and Ridge Regression Model on feature Views - label Earnings
X_views = df["Views"].to_numpy().reshape(-1,1)
y_earnings = earnings
feature = "Views"
model_comparison(X_views, y_earnings, feature)
```

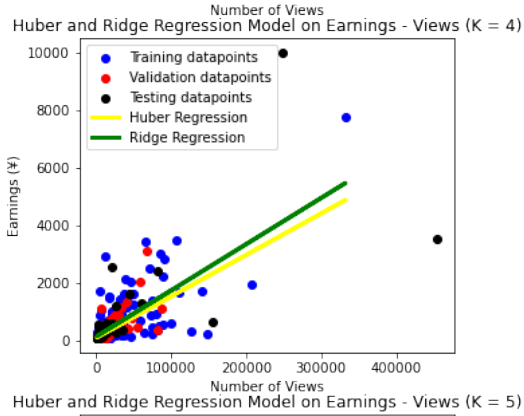
```
Huber Regression. Error measured by MSE
Slopes for each K: [0.017534499673096397, 0.01311903753088568,
0.015115781958991097, 0.014413660208038399, 0.021387415341738082]
Slope by the best model predictor: 0.014413660208038399
Intercepts for each K: [68.85232665236278, 91.50678268329034, 78.10118715618137,
88.05356873260702, 3.4000506344851755e-06]
Intercept by the best model predictor: 88.05356873260702
Training error for each K: [165997.6972830831, 164639.21833958148,
149925.02719566852, 188288.93766179125, 201386.59896352643]
Average training error: 174047.49588873016
Training error by the best model predictor: 188288.93766179125
Validation error for each K: [176166.82710417057, 240409.6608337491,
254023.99257064867, 109014.11301330589, 129830.72012304871]
```

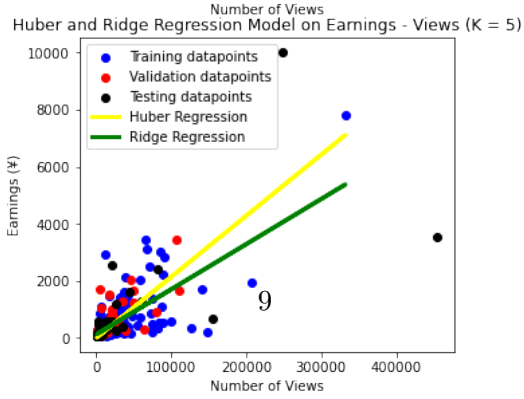
```
Average validation error: 181889.0627289846
Validation error by the best model predictor: 109014.11301330589
Testing error by the best model predictor: 1218286.7543276625

Ridge Regression. Error measured by MSE
Slopes for each K: [0.0171573198317369, 0.012258394513604803,
0.018633497069664554, 0.01606432289145588, 0.01580595472274676]
Slope by the best model predictor: 0.01606432289145588
Intercepts for each K: [121.5514030370453, 183.7844134781017, 92.3115005309242,
131.12396284600845, 123.20473956329391]
Intercept by the best model predictor: 131.12396284600845
Training error for each K: [163709.50433667115, 157965.31291004678,
136631.30400259345, 181006.80443436874, 174647.73581937424]
Average validation error: 192604.2658113158
Training error by the best model predictor: 181006.80443436874
Validation error for each K: [175587.54915896975, 253358.83209024803,
301786.5804571387, 102863.12358040737, 129425.24376981519]
Average training error: 162792.13230061086
Validation error by the best model predictor: 102863.12358040737
Testing error by the best model predictor: 1218836.3312658165
```


Huber and Ridge Regression Model on Earnings - Views (K = 1)
Training datapoints
Validation datapoints
Testing datapoints
Huber Regression
Ridge Regression
Earnings (¥)
Number of Views
0
2000
4000
6000
8000
10000
0
100000
200000
300000
400000


Huber and Ridge Regression Model on Earnings - Views (K = 2)
Training datapoints
Validation datapoints
Testing datapoints
Huber Regression
Ridge Regression
Earnings (¥)
Number of Views
0
2000
4000
6000
8000
10000
0
100000
200000
300000
400000

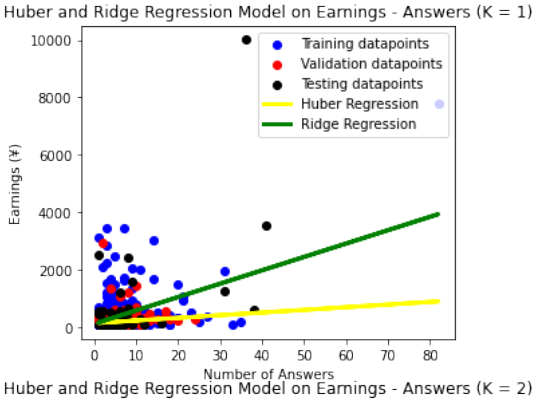

Huber and Ridge Regression Model on Earnings - Views (K = 3)
Training datapoints
Validation datapoints
Testing datapoints
Huber Regression
Ridge Regression
Earnings (¥)
Number of Views
0
2000
4000
6000
8000
10000
0
100000
200000
300000
400000

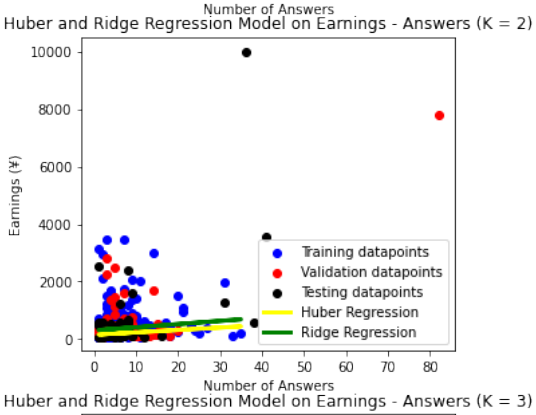

Huber and Ridge Regression Model on Earnings - Views (K = 4)
Training datapoints
Validation datapoints
Testing datapoints
Huber Regression
Ridge Regression
Earnings (¥)
Number of Views
0
2000
4000
6000
8000
10000
0
100000
200000
300000
400000

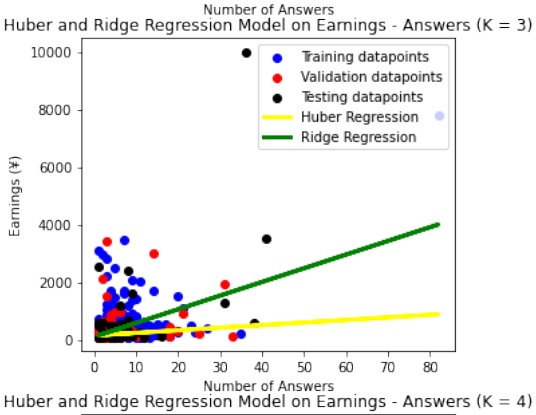

Huber and Ridge Regression Model on Earnings - Views (K = 5)
Training datapoints
Validation datapoints
Testing datapoints
Huber Regression
Ridge Regression
Earnings (¥)
Number of Views
0
2000
4000
6000
8000
10000
0
100000
200000
300000
400000

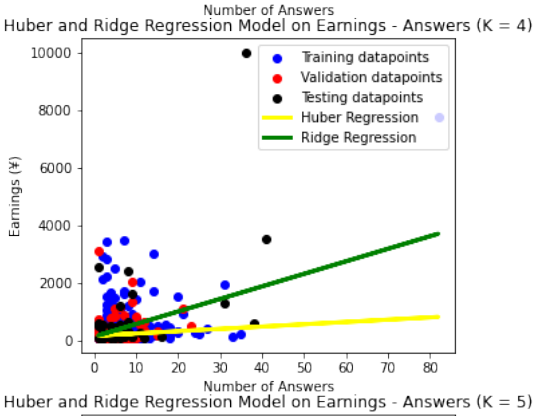
[10]:
```
# Comparing Huber and Ridge Regression Model on feature Answers - label Earnings
X_views = df["Answers"].to_numpy().reshape(-1,1)
y_earnings = earnings
feature = "Answers"
model_comparison(X_views, y_earnings, feature)
```

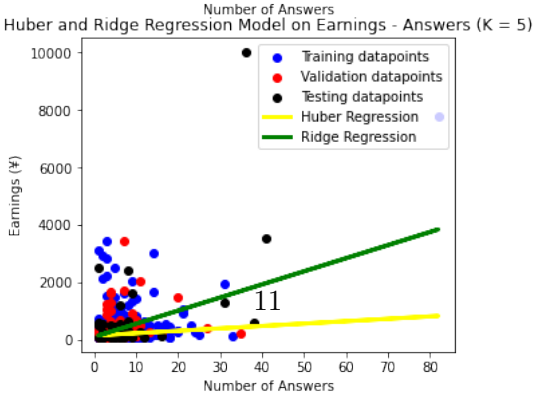
```
Huber Regression. Error measured by MSE
Slopes for each K: [9.249217372014055, 8.33326511888856, 9.000007097274038,
8.000000246814084, 8.333330674895905]
Slope by the best model predictor: 9.249217372014055
Intercepts for each K: [149.27113039436992, 151.33469762224425,
146.99999290002987, 157.9999981617545, 151.33338598430547]
Intercept by the best model predictor: 149.27113039436992
Training error for each K: [420880.2791054908, 256788.19490515735,
379984.37003062177, 410932.62718359794, 386950.34381525335]
Average training error: 371107.1630080242
Training error by the best model predictor: 420880.2791054908
Validation error for each K: [161739.34590453136, 835012.422947631,
335115.2390005152, 213800.0334341631, 314364.765473144]
Average validation error: 372006.36135199695
Validation error by the best model predictor: 161739.34590453136
Testing error by the best model predictor: 2305027.2499265973

Ridge Regression. Error measured by MSE
Slopes for each K: [46.48828481113721, 11.006621226272134, 47.555121620081245,
43.60082165370994, 45.56044607712424]
Slope by the best model predictor: 46.48828481113721
Intercepts for each K: [128.25341878802595, 302.23063670157165,
106.57322958863352, 131.12386538940115, 107.71064447955666]
Intercept by the best model predictor: 128.25341878802595
Training error for each K: [326376.171044674, 228865.91201221952,
291377.4882535789, 322616.7313831997, 300529.9388038214]
Average validation error: 357874.48778450646
Training error by the best model predictor: 326376.171044674
Validation error for each K: [198376.3666451799, 745544.2611290354,
339118.0451106158, 207584.66971260757, 298749.0963250937]
Average training error: 293953.2482994987
Validation error by the best model predictor: 198376.3666451799
Testing error by the best model predictor: 1705803.9433888039
```


Huber and Ridge Regression Model on Earnings - Answers (K = 1)
Training datapoints
Validation datapoints
Testing datapoints
Huber Regression
Ridge Regression
Earnings (¥)
Number of Answers


Huber and Ridge Regression Model on Earnings - Answers (K = 2)
Training datapoints
Validation datapoints
Testing datapoints
Huber Regression
Ridge Regression
Earnings (¥)
Number of Answers


Huber and Ridge Regression Model on Earnings - Answers (K = 3)
Training datapoints
Validation datapoints
Testing datapoints
Huber Regression
Ridge Regression
Earnings (¥)
Number of Answers


Huber and Ridge Regression Model on Earnings - Answers (K = 4)
Training datapoints
Validation datapoints
Testing datapoints
Huber Regression
Ridge Regression
Earnings (¥)
Number of Answers


Huber and Ridge Regression Model on Earnings - Answers (K = 5)
Training datapoints
Validation datapoints
Testing datapoints
Huber Regression
Ridge Regression
Earnings (¥)
Number of Answers

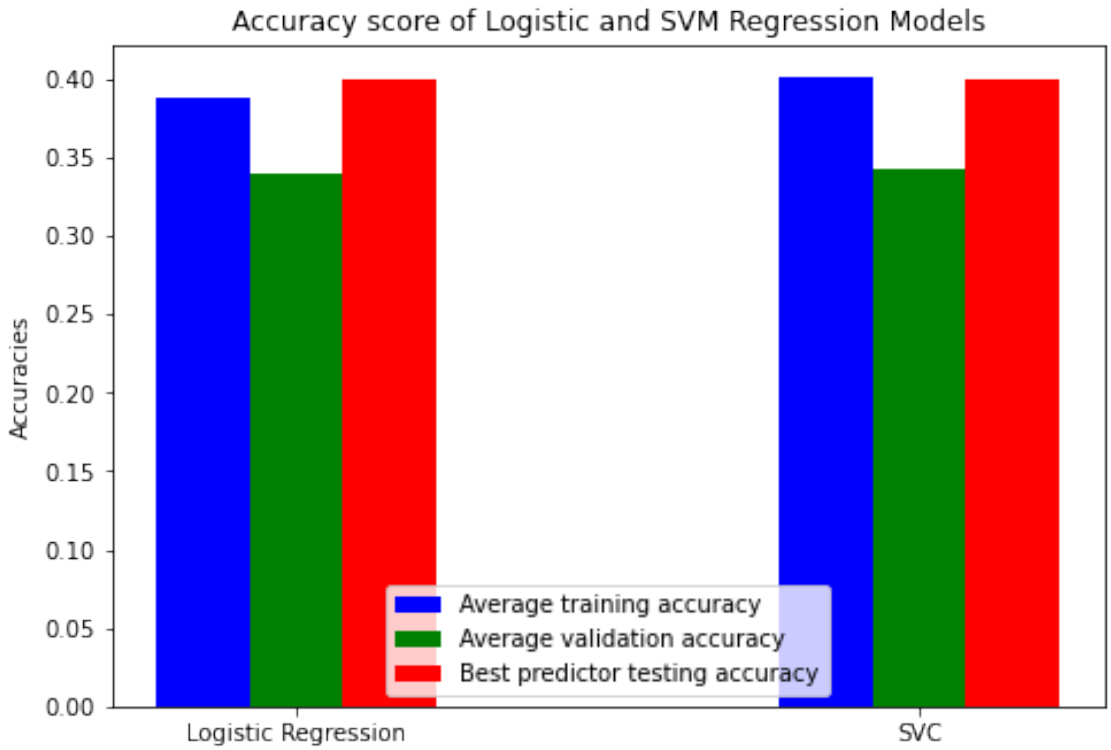
[15]:
```
copydf = df.copy()
# copydf['Topic'] = copydf['Topic'].map({"Social / News": 0, 'Health /␣
 ↪Psychology': 1, 'Culture / Politics': 2, 'Entertainment / Hobbies': 3, 'Life␣
 ↪/ Habit': 4, 'Technology / Internet': 5, 'Business / Career': 6, 'Love /␣
 ↪Relationship': 7, 'Education / Science': 8, 'Beauty / Fashion': 9})
copydf['Topic'] = copydf['Topic'].map({"Social / News": 0, 'Health /␣
 ↪Psychology': 1, 'Culture / Politics': 0, 'Entertainment / Hobbies': 2, 'Life␣
 ↪/ Habit': 1, 'Technology / Internet': 2, 'Business / Career': 0, 'Love /␣
 ↪Relationship': 1, 'Education / Science': 2, 'Beauty / Fashion': 2})
X_topics = copydf["Topic"].to_numpy()
y_earnings = earnings.reshape(-1,1)

tr_acc = { "Logistic": [], "Svm": [] }
val_acc = { "Logistic": [], "Svm": [] }
predictors = { "Logistic": [], "Svm": [] }

X_rem, X_test, y_rem, y_test = train_test_split(y_earnings, X_topics,␣
 ↪train_size=0.9, random_state = 42)
K = 5  # Number of splits
kf = KFold(n_splits=K, shuffle=True, random_state=42)    # Create a KFold␣
 ↪object with 'K' splits
i = 0
for train_indices, val_indices in kf.split(X_rem):
    X_train = X_rem[train_indices,:]    # Get the training set
    X_val = X_rem[val_indices,:]    # Get the validation set
    y_train = y_rem[train_indices]    # Get the training set
    y_val = y_rem[val_indices]    # Get the validation set

    # for Logistic Regression model training
    LogisticModel = LogisticRegression(max_iter = 1000)
    fitLogisticModel = LogisticModel.fit(X_train, y_train)
    predictors["Logistic"].append(fitLogisticModel)
    y_pred_train_Logistic = fitLogisticModel.predict(X_train)
    # for Logistic Regression validation
    y_pred_val_Logistic = fitLogisticModel.predict(X_val)
    tr_acc["Logistic"].append(accuracy_score(y_train, y_pred_train_Logistic))
    val_acc["Logistic"].append(accuracy_score(y_val, y_pred_val_Logistic))

    # for SVM Regression model training
    SvmModel = SVC(max_iter = 1000)
    fitSvmModel = SvmModel.fit(X_train, y_train)
    predictors["Svm"].append(fitSvmModel)
    y_pred_train_Svm = fitSvmModel.predict(X_train)
    # for SVM Regression validation
```

```
    y_pred_val_Svm = fitSvmModel.predict(X_val)
    tr_acc["Svm"].append(accuracy_score(y_train, y_pred_train_Svm))
    val_acc["Svm"].append(accuracy_score(y_val, y_pred_val_Svm))

best_predictor_index_Logistic = val_acc["Logistic"].
 ↪index(max(val_acc["Logistic"]))
best_predictor_index_Svm = val_acc["Svm"].index(max(val_acc["Svm"]))
y_pred_test_Logistic = predictors["Logistic"][best_predictor_index_Logistic].
 ↪predict(X_test)
y_pred_test_Svm = predictors["Svm"][best_predictor_index_Svm].predict(X_test)

acc_train_Logistic = sum(tr_acc["Logistic"])/len(tr_acc["Logistic"])
acc_val_Logistic = sum(val_acc["Logistic"])/len(val_acc["Logistic"])
acc_test_Logistic = accuracy_score(y_test, y_pred_test_Logistic)
print("Logistic Regression. Accuracy score measured by accuracy_score")
print(f'Training accuracy for each K: {tr_acc["Logistic"]}')
print(f'Validation accuracy for each K: {val_acc["Logistic"]}')
print(f'Average training accuracy: {acc_train_Logistic}')
print(f'Average validation accuracy: {acc_val_Logistic}')
print(f'Testing score by the best model predictor: {acc_test_Logistic}\n')

acc_train_Svm = sum(tr_acc["Svm"])/len(tr_acc["Svm"])
acc_val_Svm = sum(val_acc["Svm"])/len(val_acc["Svm"])
acc_test_Svm = accuracy_score(y_test, y_pred_test_Svm)
print("SVM Regression. Accuracy score measured by accuracy_score")
print(f'Training accuracy for each K: {tr_acc["Svm"]}')
print(f'Validation accuracy for each K: {val_acc["Svm"]}')
print(f'Average training accuracy: {acc_train_Svm}')
print(f'Average validation accuracy: {acc_val_Svm}')
print(f'Testing score by the best model predictor: {acc_test_Svm}\n')

earnings_test = np.arange(0, 10001, 20)
earnings_predict = predictors["Svm"][best_predictor_index_Logistic].
 ↪predict(earnings_test.reshape(-1,1))
np.set_printoptions(threshold = np.inf)
print(earnings_predict)
# plot bar results
acc_Logistic = list[acc_train_Logistic, acc_val_Logistic, acc_test_Logistic]
acc_Svm = list[acc_train_Svm, acc_val_Svm, acc_test_Svm]
acc_train = [acc_train_Logistic, acc_train_Svm]
acc_val = [acc_val_Logistic, acc_val_Svm]
acc_test = [acc_test_Logistic, acc_test_Svm]
X = np.arange(2)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(X - 0.15, acc_train, color = 'b', width = 0.15, label = "Average␣
 ↪training accuracy")
```

```
ax.bar(X, acc_val, color = 'g', width = 0.15, label = "Average validation␣
 ↪accuracy")
ax.bar(X + 0.15, acc_test, color = 'r', width = 0.15, label = "Best predictor␣
 ↪testing accuracy")
ax.set_xticks(X, labels = ("Logistic Regression","SVC"))
#ax.set_xticklabels(["LogisticRegression","SVC"])
ax.set_ylabel("Accuracies")
ax.set_title("Accuracy score of Logistic and SVM Regression Models")
ax.legend(loc="lower center")
```

```
Logistic Regression. Accuracy score measured by accuracy_score
Training accuracy for each K: [0.40555555555555556, 0.37222222222222223, 0.4,
0.38333333333333336, 0.38055555555555554]
Validation accuracy for each K: [0.3111111111111111, 0.43333333333333335,
0.3333333333333333, 0.3888888888888889, 0.23333333333333334]
Average training accuracy: 0.3883333333333333
Average validation accuracy: 0.33999999999999997
Testing score by the best model predictor: 0.4

SVM Regression. Accuracy score measured by accuracy_score
Training accuracy for each K: [0.40555555555555556, 0.3972222222222222,
0.40555555555555556, 0.3861111111111111, 0.41388888888888886]
Validation accuracy for each K: [0.3111111111111111, 0.36666666666666664,
0.3111111111111111, 0.3888888888888889, 0.3333333333333333]
Average training accuracy: 0.40166666666666667
Average validation accuracy: 0.3422222222222222
Testing score by the best model predictor: 0.4

[1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
[15]: <matplotlib.legend.Legend at 0x7f96468653d0>
```


Accuracy score of Logistic and SVM Regression Models
Accuracies
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Average training accuracy
Average validation accuracy
Best predictor testing accuracy
Logistic Regression
SVC