
Do we need more bikes?

Project in Statistical Machine Learning

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 In this project we develop, and study different statistical machine learning models
2 for predicting whether the number of available bikes at a given hour should be
3 increased, a project by the District Department of Transportation in Washington
4 D.C. The training data set consists of 1600 instances of hourly bike rentals, and
5 a test set of 400 instances. The models for prediction we have used are: *Logistic*
6 *regression*, *Discriminant methods: LDA, QDA, k- Nearest Neighbour*, and *Tree*
7 *Based Methods*. We have found that THE MODEL gives best prediction, with
8 accuracy ??????

9 1 Plan

10 1.1 From Intro

- 11 (i) Explore and preprocess data
- 12 (ii) try some or all classification methods, which are these?
 - 13 • Logistic Regression
 - 14 • Discriminant analysis: LDA, QDA
 - 15 • K-nearest neighbor
 - 16 • Tree-based methods: classification trees, random forests, bagging
 - 17 • Boosting
- 18 (iii) Which of these are to be "put in production"?

19 1.2 From Data analysis task

- 20 • Can any trend be seen comparing different hours, weeks, months?
- 21 • Is there any difference between weekdays and holidays?
- 22 • Is there any trend depending on the weather?

23 1.3 From Implementation of methods

24 Each group member should implement one family each, who did what shall be clear!

25 DNNs are encouraged to be implemented, do this if there is time. (DNN is not a thing a group member can claim as their family.)

26 Implement a naive version, let's do: *Always low_bike_demand*

28 1.3.1 What to do with each method

- 29 1. Implement the method (each person individually)
- 30 2. Tune hyper-parameters, discuss how this is done (each person individually)
- 31 3. Evaluate with for example cross-validation. Don't use E_{k-fold} (what is that?) (need to do
- 32 together)
- 33 4. (optional) Think about input features, are all relevant? (together)

34 Before training, unify pre-processing FOR ALL METHODS and choose ONE OR MULTIPLE

35 metrics to evaluate the model. (is it necessary to have the same for all?, is it beneficial?) Examples:

- 36 • accuracy
- 37 • f1-score
- 38 • recall
- 39 • precision

40 Use same test-train split for ALL MODELS

41 2 Introduction

42 Statistical machine learning is a subject that aims to build and train algorithms, that analyse large

43 amount of data, and make predictions for the future, which are computed by using established

44 statistical models, and tools from functional analysis. This is a project in supervised, statistical

45 machine learning, where several models were created, and trained, in order to analyse which one of

46 them gives best prediction for the project "Do we need more bikes", where we want to understand,

47 and predict if there is a high, or low demand of city bikes in the public transportation of Washington,

48 a project by the District Department of Transportation in Washington D.C..

49 The data set used for training our models, consist of 15 variables, containing quantitative/qualitative

50 data. We developed several models, and evaluated them with cross-validation, in order to understand

51 which algorithm gives the best prediction.

52 3 Theoretical Background

53 3.1 Mathematical Overview of the Models

54 3.1.1 Logistic Regression

55 The backbone of logistic regression is linear regression, i.e. finding the least-squares solution to an
56 equation system

$$X\theta = y \quad (1)$$

57 given by the normal equations

$$X^T X \theta = X^T y \quad (2)$$

58 where X is the training data matrix, θ is the coefficient vector and b is the training output. The
59 parameter vector is then used in the sigmoid function:

$$\sigma(z) = \frac{e^z}{1 + e^z} : \mathbb{R} \rightarrow [0, 1], \quad (3)$$

$$z = x^T \theta, \quad (4)$$

60 where x is the testing input. This gives a statistical interpretation of the input vector. In the case of a
61 binary True/False classification, the value of the sigmoid function then determines the class.

62 3.1.2 Random forest

63 The random forest method is a based upon decision trees, i.e. dividing the data point into binary
64 groups based on Gini-impurity, entropy or classification error, Gini being the most common. These
65 divisions are then used to create a binary tree shown in figure ??Tree) and where the leaf-nodes
66 are used to classify the target variables based on the input. As of itself the disition tree tends to
67 have unsatisfying results which leads to methodes like random forest and sandbagging that boost its
68 accuracy. Sandbagging is a way to sampel the data in order to get multiple datasets from the same
69 data. One then creates a desition-tree for every subset data to then combine them into one model. This
70 lessens the variance of the model but increases bias. This means that sandbagging can increase false
71 negatives which in theis aplication makes i nonviable. Random forest on the otherhand is viable, it
72 creates mutple trees whilse disrecarding random input variable this randomnes decreases overfitting
73 creating a more robust model.

74 3.1.3 Non-parametric method: k-Nearest Neighbour

75 *k-Nearest Neighbour* (k -NN) is a distance based method that takes a k amount of points from the
76 training data set, called *neighbours*, computes the distance between them, then assumes that the
77 predicted value $\hat{y}(x_*)$ follows the trend of the k - nearest neighbours. Since k -NN uses the training
78 data explicitly it is also called a *nonparametric* method.

79 The k -NN method can be divided into several subcategories, inter alias *classification* k -NN method,
80 *regression* k -NN method. In this project, we are using the classification method, since we are trying
81 to predict in which of the two classes low, or high demand, the given, and predicted data points
82 belong.

83 The classification k -NN algorithm evaluates $\hat{y}(x_*)$ by computing the most frequently occurring class
84 among the k nearest neighbours. Here, we try to identify whether a data point belong to the high
85 demand-class. Denote c = high demand class. For simplicity, assume Euclidean ambience. Then

$$\hat{y}(x_*) = \arg \max_c \sum_{n \in \mathbb{N}} \chi_{(y_n=c)},$$

86 where y_i is the class of the nearest neighbour, χ is the characteristic function

$$\chi_{(y_i=c)} = \begin{cases} 1 & \text{if } y_n = c, \\ 0 & \text{otherwise.} \end{cases}$$

87 It is very common to use a weighted sum to predict the next value, i.e.

$$\hat{y}(x_*) = \arg \max_c \sum_{n \in \mathbb{N}} \frac{\chi_{(y_n=c)}}{d(x, x_n)},$$

where d is the standard Euclidean metric, computing the distance between an input x , and a neighbour x_n .

When using this model it is important to choose an optimal k -value. There are several tests for this, here we implement *uniform weighting*, and *distance weighting*. The first algorithm creates a k -NN model for each new $k \in [1, 500]$, and trains the model with uniform weights, i.e. the contribution of all neighbours is equal. Similarly, the latter trains a k -NN classifier for each $k \in [1, 500]$, with the difference that it uses distance based weighting, i.e. closer neighbours have greater influence. After testing different upper boundaries for k , the two models gave good results in the interval $[1, 500]$, see Figure 1. From the figures, we can see that the second test gives a better value for k , since the plot follows smoother trend, in comparison to the uniform weighting test, which makes it easier to identify an optimal k value ($k = 120$). Moreover, the distance weighting algorithm is providing results for larger values of k , that is for $k \in [1, 400)$ before the curve converges, while the uniform weighting algorithm converges earlier, when $k = 120$. This means that for large k , both test algorithms make prediction based on the most common class in the data set, instead of making prediction based on the behaviour of the neighbours. Thus for sufficiently large k , for any given data point, the model will consider unnecessarily large amount of neighbours, and the prediction will be evaluated to belong to the most frequent class. Since the distance weighting has a larger range of k -value, it should be more trustworthy.

When $k = 120$, the accuracy of the model is 92%.

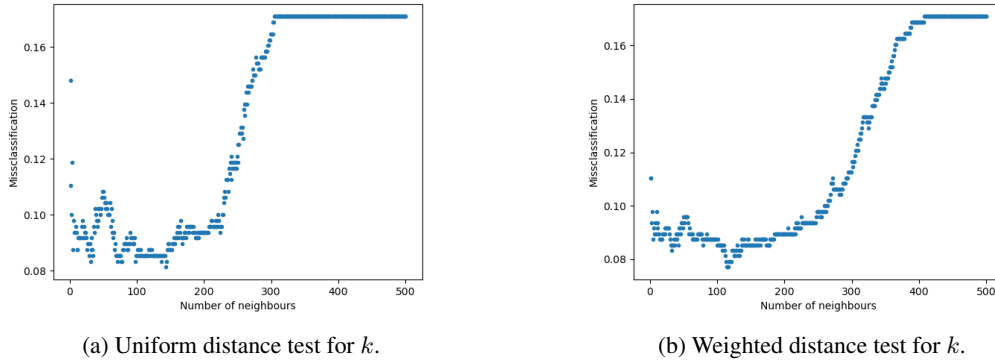


Figure 1: Test for choosing an optimal k -value.

3.1.4 Discriminant analysis: LDA and QDA

Linear Discriminant Analysis is a generative model, which means it is a model that's creating and using a probability distribution $P(\mathbf{x}, y)$ to create an estimation for the probability $P(y = m|\mathbf{x})$ using Bayes theorem.

Bayes theorem is:

$$p(y|\mathbf{x}) = \frac{p(y, \mathbf{x})}{p(\mathbf{x})} = \frac{p(y)p(\mathbf{x}|y)}{\int_y p(y, \mathbf{x})}$$

For the discrete version it is obtained:

$$p(y = m|\mathbf{x}) = \frac{p(y = m)p(\mathbf{x}|y = m)}{\sum_{m=1}^M p(y = m)p(\mathbf{x}|y = m)}$$

For this form of the equation to be useful, it is necessary to obtain an accurate estimation of $p(y = m)$ and $p(\mathbf{x}|y = m)$ for all classes m .

In LDA, $p(y = m)$ is estimated by counting the percentage of data points (in the training data) being in each of the classes and using that percentage as the probability of a data point being in that class.

In mathematical terms:

$$p(y = m) = \frac{1}{n} \sum_{i=1}^n \mathbb{I}\{y_i = m\} = \frac{n_m}{n}$$

118 To estimate the probability distribution $p(\mathbf{x}|y = m)$, a multi-dimensional gaussian distribution is
 119 used:

$$\mathcal{N}(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1}(\mathbf{x} - \mu)\right)$$

120 Where \mathbf{x} is the d-dimensional data point, μ is the (d-dimensional) mean of the random variable. Σ is
 121 the symmetric, positive definite covariance matrix defined by:

$$\Sigma = \frac{1}{n - M} \sum_{m=1}^M \sum_{i:y_i=m} (\mathbf{x}_i - \mu_m)(\mathbf{x}_i - \mu_m)^T$$

122 Using these estimations results in an expression for the quantity $p(y = m|\mathbf{x})\forall m$. LDA then uses
 123 maximum likelihood to categorize an input \mathbf{x} into a class m .

124
 125 Quadratic discriminant analysis (QDA) is heavily based of LDA with the sole difference
 126 being how the covariance matrix Σ is created. In LDA, the covariance matrix is assumed to be the
 127 same for data in each and every class. In QDA however, the covariance matrix is calculated for each
 128 class as follows:

$$\Sigma_m = \frac{1}{n_m - 1} \sum_{i:y_i=m} (\mathbf{x}_i - \mu_m)(\mathbf{x}_i - \mu_m)^T$$

129 One thing to note about LDA and QDA is that the use of a multi-variable gaussian distribution
 130 benefits normally distributed variables. In this project however, there is a dependence on positive
 131 definite values which are not normally distributed by nature. This is an issue when using QDA since
 132 in the class of *high_bike_demand*, all data points have a snow depth of 0 and has hence no variance.
 133 This results in this class having a undefined inverse for the covariance matrix. The solution used was
 134 to exclude this variable from this model.

135 3.2 Input Data Modification

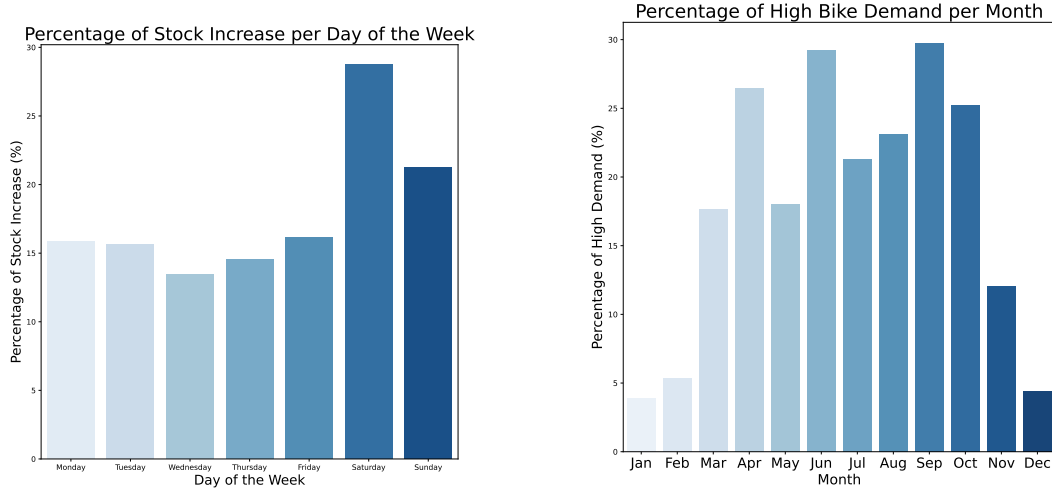
136 By plotting the data and analyzing the .csv file, some observations were made. The different inputs
 137 were then changed accordingly:

- 138 • *Kept as-is:* weekday, windspeed, visibility, temp
- 139 • *Modified:*
 - 140 – month - split into two inputs, one cosine and one sine part. This make the new inputs
 - 141 linear and can follow the fluctuations of the year. The original input was discarded.
 - 142 – hour_of_day - split into three boolean variables: demand_day, demand_evening,
 - 143 and demand_night, reflecting if the time was between 08-14, 15-19 or 20-07 respec-
 - 144 tively. This was done because plotting the data showed three different plateaus of
 - 145 demand for the different time intervals. The original input was discarded.
 - 146 – snowdepth, precip were transformed into booleans, reflecting if it was raining or
 - 147 if there was snow on the ground or not. This was done as there was no times where
 - 148 demand was high when it was raining or when there was snow on the ground.
- 149 • *Removed:* cloudcover, day_of_week, snow, dew, holiday, summertime. These were
- 150 removed due to being redundant (e.g. summertime), not showing a clear trend (e.g.
- 151 cloudcover), giving a worse score when used, or all three (e.g. day_of_week).

152 4 Data Analysis

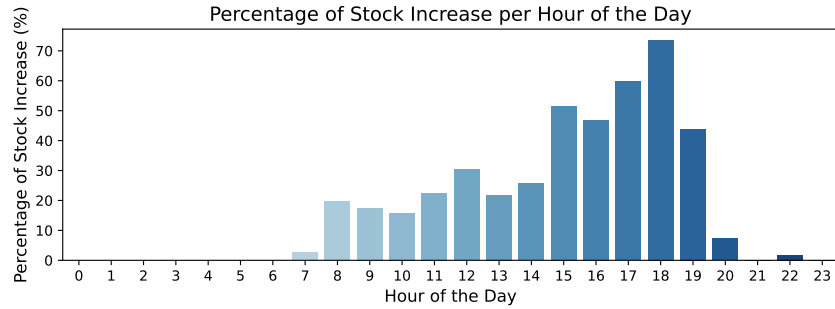
153 In the given data, there are some numerical and categorical features:

- 154 • *Numerical:* temp, dew, humidity, precip, snow, snowdepth, windspeed, cloudcover
- 155 and visibility.
- 156 • *Categorical:* hour_of_day, day_of_week, month, holiday, weekday, summertime, and
- 157 increase_stock



(a) Demand per day of week.

(b) Demand per month.



(c) Demand per hour of day.

Figure 2: Bike demand vs. day of week and month.

There are some trends seen in the data when it comes to time and weather. From figure 2, one can see a periodic relationship for the months, where there is a higher demand during the warmer months, loosely following a trigonometric curve. Over the week, the demand is rather stable, with a peak on the weekend, especially Saturdays.

Looking at the weather (figure 3); if there is rain or if there is snow on the ground, there is close to always low demand. Cloudcover did not make a big impact, which is also intuitive, as a cloudy day does not make biking more difficult. Dew point also does not have a clear trend, while humidity however has a clear trend downwards as the humidity increases. Temperature had a more clear impact, where more people wanted to bike the warmer it got.

The overall trend is that about one eighth of observations correspond to a high bike demand. During the night, or in bad weather, the demand is (intuitively) low. But during rush hour (figure 2c), the demand is very high, and should probably be increased in order to minimize excessive CO₂ emissions.

5 Result

The method used to evaluate the different models were simply chosen to be the accuracy defined by:

$$\text{accuracy} = \frac{n_{\text{correct}}}{n_{\text{tot}}}$$

The different models were tested and the accuracy where:

Here you can clearly see random forest and k-nearest neighbour are the best classifiers both outperforming linear and quadratic regression on accuracy, precision and recall. Out of random forest and kNN the group would proceed with the kNN method, its higher accuracy and precision score out

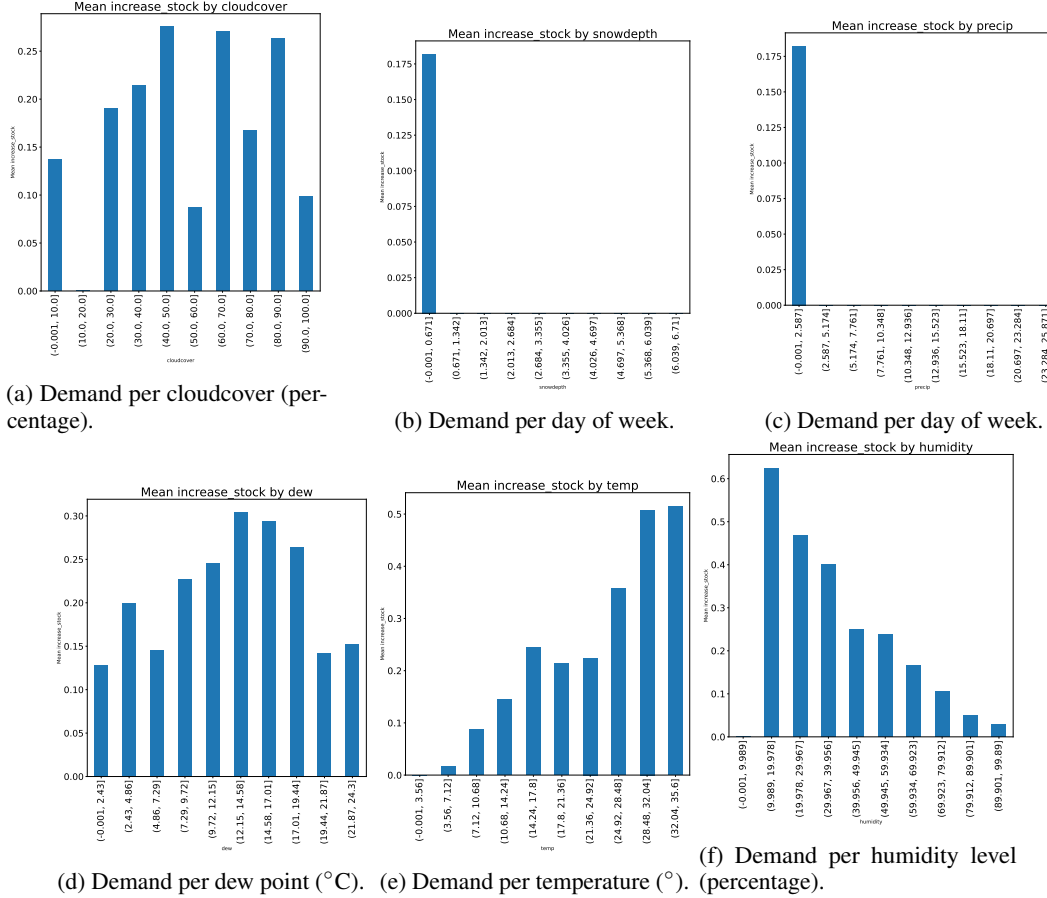


Figure 3: Bike demand vs. various weather parameters.

Accuracy of the models

Model	Accuracy	Precision	Recall
LDA	85%	53%	50%
QDA	87%	67%	36%
k-nearest neighbour	92%	81%	70%
Random Forest	91%	77%	71%

176 waying the slightly better recall score of random forest. This will mean a slight loss in income caused
 177 by increasing false negatives but is thought to be covered by fewer false positives.

178 A Appendix

```

179 1 import pandas as pd
180 2 import numpy as np
181 3 from sklearn.model_selection import train_test_split
182 4 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
183 5 from sklearn.linear_model import LogisticRegression
184 6 from sklearn.metrics import accuracy_score
185 7 from sklearn.metrics import classification_report
186 8
187 9 df = pd.read_csv('training_data_vt2025.csv')
188 10
189 11 # modify the month to represent the periodicity that is observed in
190 12 data.
191 13 df['month_cos'] = np.cos(df['month']*2*np.pi/12)
192 14 df['month_sin'] = np.sin(df['month']*2*np.pi/12)
193 15
194 16 # time of day, replaced with 3 bool values: is_night, is_day and
195 17 is_evening,
196 18 # adding the new categories back in the end.
197 19 def categorize_demand(hour):
198 20     if 20 <= hour or 7 >= hour:
199 21         return 'night'
200 22     elif 8 <= hour <= 14:
201 23         return 'day'
202 24     elif 15 <= hour <= 19:
203 25         return 'evening'
204 26
205 27 df['time_of_day'] = df['hour_of_day'].apply(categorize_demand)
206 28 df_dummies = pd.get_dummies(df['time_of_day'], prefix='is', drop_first
207 29 =False)
208 30 df = pd.concat([df, df_dummies], axis=1)
209 31
210 32 # Create bool of snowdepth and percipitation
211 33 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
212 34 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
213 35
214 36 # Seperate training data from target
215 37 X=df[['#holiday',
216 38     'weekday',
217 39     '#summertime',
218 40     'temp',
219 41     '#dew',
220 42     '#humidity',
221 43     '#visibility',
222 44     '#windspeed',
223 45     '#month',
224 46     'month_cos',
225 47     'month_sin',
226 48     '#hour_of_day',
227 49     'is_day',
228 50     'is_evening',
229 51     'is_night',
230 52     '#hour_cos',
231 53     '#hour_sin',
232 54     'snowdepth_bool',
233 55     'precip_bool'
234 56 ]]
235 57
236 58 y=df['increase_stock']
237 59
238 60 # Split dataset into training and test sets
239 61 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
240 62 =0.2, random_state=42)
241 63

```



```

24250 # Apply Linear Discriminant Analysis (LDA)
24351 lda = LinearDiscriminantAnalysis(n_components=1)
24452 X_train_lda = lda.fit_transform(X_train, y_train)
24553 X_test_lda = lda.transform(X_test)
24654
24755 # Train a classifier (Logistic Regression)
24856 clf = LogisticRegression()
24957 clf.fit(X_train_lda, y_train)
25058
25159 # Make predictions
25260 y_pred = clf.predict(X_test_lda)
25361
25462 # Evaluate accuracy
25563 accuracy = accuracy_score(y_test, y_pred)
25664 print(f"Model Accuracy: {accuracy:.2f}")
25765
25866 print(classification_report(y_test, y_pred))

```

Listing 1: Code for LDA

```

259 1 import pandas as pd
260 2 import numpy as np
261 3 from sklearn.model_selection import train_test_split
262 4 from sklearn.discriminant_analysis import
263     QuadraticDiscriminantAnalysis
264 5 from sklearn.metrics import accuracy_score
265 6 from sklearn.metrics import classification_report
266 7
267 8 df = pd.read_csv('training_data_vt2025.csv')
268 9
26910 # modify the month to represent the periodicity that is observed in
270     data.
27111 df['month_cos'] = np.cos(df['month']*2*np.pi/12)
27212 df['month_sin'] = np.sin(df['month']*2*np.pi/12)
27313
27414 # time of day, replaced with 3 bool values: is_night, is_day and
275     is_evening,
27615 # adding the new categories back in the end.
27716 def categorize_demand(hour):
27817     if 20 <= hour or 7 >= hour:
27918         return 'night'
28019     elif 8 <= hour <= 14:
28120         return 'day'
28221     elif 15 <= hour <= 19:
28322         return 'evening'
28423
28524 df['time_of_day'] = df['hour_of_day'].apply(categorize_demand)
28625 df_dummies = pd.get_dummies(df['time_of_day'], prefix='is', drop_first
287     =False)
28826 df = pd.concat([df, df_dummies], axis=1)
28927
29028 # Create bool of snowdepth and percipitation
29129 df['snowdepth_bool'] = df['snowdepth'].where(df['snowdepth'] == 0, 1)
29230 df['precip_bool'] = df['precip'].where(df['precip'] == 0, 1)
29331
29432 # Seperate training data from target
29533 X=df[['#holiday',
29634     'weekday',
29735     '#summertime',
29836     'temp',
29937     '#dew',
30038     '#humidity',
30139     '#visibility',
30240     '#windspeed',
30341     '#month',

```

```

30412         'month_cos',
30513         'month_sin',
30614         #'hour_of_day',
30715         'is_day',
30816         'is_evening',
30917         'is_night',
31018         #'snowdepth_bool',
31119         'precip_bool'
31220     ]]
31321
31422 y=df['increase_stock']
31523
31624 # Split dataset into training and test sets
31725 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
318         =0.2, random_state=42)
31926
32027 # Apply Quadratic Discriminant Analysis (QDA)
32128 qda = QuadraticDiscriminantAnalysis()
32229 X_train_lda = qda.fit(X_train, y_train)
32330
32431 # Make predictions
32532 y_pred = qda.predict(X_test)
32633
32734 # Evaluate accuracy
32835 accuracy = accuracy_score(y_test, y_pred)
32936 print(f"Model Accuracy: {accuracy:.2f}")
33037
33138 print(classification_report(y_test, y_pred))

```

Listing 2: Code for QDA

```

332 1 import numpy as np
333 2 import pandas as pd
334 3 import matplotlib.pyplot as plt
335 4 import sklearn.linear_model as skl_lm
336 5 import sklearn.preprocessing as pp
337 6 import sklearn.metrics as skl_m
338 7
339 8 import sklearn.neighbors as skl_nb
340 9
34110 df = pd.read_csv('training_data_vt2025.csv')
34211 #df.info()
34312
34413 # Modify the dataset, emphasizing different variables
34514 #df.iloc[:,12]=df.iloc[:,12]**2
34615 #df.iloc[:,13]=np.sqrt(df.iloc[:,13])
34716 #df.iloc[:,11] = df.iloc[:,11]**2
34817
34918 df['month_cos'] = np.cos(df.month*np.pi/12)
35019 df['month_sin'] = np.sin(df.month*np.pi/12)
35120
35221 # time of day, replaed with low,medium and high demand,
35322 # adding the new categories back in the end.
35423 def categorize_demand(hour):
35524     if 20 <= hour or 7 >= hour:
35625         return 'night'
35726     elif 8 <= hour <= 14:
35827         return 'day'
35928     elif 15 <= hour <= 19:
36029         return 'evening'
36130
36231 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
36332 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
364         drop_first=False)
36533 df = pd.concat([df, df_dummies], axis=1)

```

```

36634
36735 # converting to bools
36836 def if_zero(data):
36937     if data == 0:
37038         return True
37139     else:
37240         return False
37341
37442 # temperature
37543
37644 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
37745 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
37846
37947 # Split into train and test:
38048
38149 #df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
38250 #df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
38351 np.random.seed(0)
38452
38553 df_modified=df[['holiday',
38654                 'weekday',
38755                 '#summertime',
38856                 'temp',
38957                 '#dew',
39058                 'humidity',
39159                 'visibility',
39260                 'windspeed',
39361                 'month_cos',
39462                 'month_sin',
39563                 'demand_day',
39664                 'demand_evening',
39765                 'demand_night',
39866                 'snowdepth_bool',
39967                 'precip_bool',
40068                 'increase_stock']]
40169
40270 N = df_modified.shape[0]
40371 n = round(0.7*N)
40472 trainI = np.random.choice(N,size=n,replace=False)
40573 trainIndex = df_modified.index.isin(trainI)
40674 train = df_modified.iloc[trainIndex]
40775 test = df_modified.iloc[~trainIndex]
40876
40977 # Set up X,Y
41078
41179 # Train data
41280 X = train.iloc[:,0:-2]
41381 Y = train['increase_stock']
41482
41583 # Test data
41684 X_test = test.iloc[:,0:-2]
41785 Y_test = test['increase_stock']
41886
41987
42088 """
42189 # Tests for k-value
42290 # TEST 1 - uniform distance
42391 missclassification = []
42492 for k in range(500): # Try n_neighbours = 1, 2, ...,
42593
42694     #kNN method
42795     scaler = pp.StandardScaler().fit(X)
42896     model = skl_nb.KNeighborsClassifier(n_neighbors = k+1, weights = '
429     uniform')
43097     model.fit(scaler.transform(X),Y)

```

```

43198
43299     # Prediction
43300     y_hat = model.predict(scaler.transform(X_test))
43401     missclassification.append(np.mean(y_hat != Y_test))
43502
43603 K = np.linspace(1, 500, 500)
43704 plt.plot(K, missclassification, '.')
43805 plt.ylabel('Missclassification')
43906 plt.xlabel('Number of neighbours')
44007 plt.show()
44108
44209 #TEST 2
44300 missclassification = []
44401 for k in range(500): # Try n_neighbours = 1, 2, ...,
44502
44603     #kNN method
44704     scaler = pp.StandardScaler().fit(X)
44805     model = skl_nb.KNeighborsClassifier(n_neighbors = k+1, weights = '
449     distance')
45006     model.fit(scaler.transform(X),Y)
45107
45208     # Prediction
45309     y_hat = model.predict(scaler.transform(X_test))
45400     missclassification.append(np.mean(y_hat != Y_test))
45501
45602 K = np.linspace(1, 500, 500)
45703 plt.plot(K, missclassification, '.')
45804 plt.ylabel('Missclassification')
45905 plt.xlabel('Number of neighbours')
46006 plt.show()
46107 """
46208
46309
46400
46501 # creating the model
46602 model = skl_nb.KNeighborsClassifier(n_neighbors = 120, weights = '
467     distance')
46803
46904
47005 # Scaling the data, otherwise
47106 scaler = pp.StandardScaler().fit(X)
47207 model.fit(scaler.transform(X),Y)
47308 y_hat = model.predict(scaler.transform(X_test))
47409
47500
47601
47702 '''
47803 # oskalad data
47904 model.fit(X,Y)
48005 y_hat = model.predict(X_test)'''
48106
48207 # Get confusion matrix
48308 diff = pd.crosstab(y_hat, Y_test)
48409 print(f'Confusion matrix: \n {diff}')
48500
48601 # No. of TP,TN,FP,FN
48702 '''TP = diff.iloc[0,0]
48803 TN = diff.iloc[1,1]
48904 FP = diff.iloc[1,0]
49005 FN = diff.iloc[0,1]'''
49106
49207 # Get metrics:
49308 print(skl_m.classification_report(Y_test, y_hat))

```

Listing 3: Code for KNN

```

494 1 import pandas as pd
495 2 import numpy as np
496 3 import matplotlib
497 4 import matplotlib.pyplot as plt
498 5 from sklearn import tree
499 6 from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
500 7 import graphviz
501 8 from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
502 9 from sklearn.metrics import classification_report
503 10
504 11 df = pd.read_csv('training_data_vt2025.csv')
505 12 #df.info()
506 13
507 14 # Modify the dataset, emphasizing different variables
508 15 df.iloc[:,12]=df.iloc[:,12]**2
509 16 df.iloc[:,13]=np.sqrt(df.iloc[:,13])
510 17 df.iloc[:,11] = df.iloc[:,11]**2
511 18
512 19 df['month_cos'] = np.cos(df.month*np.pi/12)
513 20 df['month_sin'] = np.sin(df.month*np.pi/12)
514 21
515 22 # time of day, replaed with low,medium and high demand,
516 23 # adding the new categories back in the end.
517 24 def categorize_demand(hour):
518 25     if 20 <= hour or 7 >= hour:
519 26         return 'night'
520 27     elif 8 <= hour <= 14:
521 28         return 'day'
522 29     elif 15 <= hour <= 19:
523 30         return 'evening'
524 31
525 32 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
526 33 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
527 34 drop_first=False)
528 35 df = pd.concat([df, df_dummies], axis=1)
529 36
530 37 # converting to bools
531 38 def if_zero(data):
532 39     if data == 0:
533 40         return True
534 41     else:
535 42         return False
536 43
537 44 # temperature
538 45
539 46 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
540 47 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
541 48
542 49 # Split into train and test:
543 50
544 51 #df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
545 52 #df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
546 53 np.random.seed(0)
547 54
548 55 df_modified=df[['holiday',
549 56                 'weekday',
550 57                 '#summertime',
551 58                 'temp',
552 59                 '#dew',
553 60                 '#humidity',
554 61                 'visibility',
555 62                 'windspeed',
556 63                 'month_cos',
557 64                 'month_sin',
558 65                 'demand_day',

```

```

55955         'demand_evening',
56056         'demand_night',
56167         'snowdepth_bool',
56258         'precip_bool',
56359         'increase_stock']]
56470
56571 N = df_modified.shape[0]
56672 n = round(0.7*N)
56773 trainI = np.random.choice(N,size=n,replace=False)
56874 trainIndex = df_modified.index.isin(trainI)
56975 train = df_modified.iloc[trainIndex]
57076 test = df_modified.iloc[~trainIndex]
57177
57278 X_train = train.drop(columns=['increase_stock'])
57379 # Need to transform the qualitative variables to dummy variables
57480
57581 y_train = train['increase_stock']
57682
57783 model = RandomForestClassifier(random_state=42)
57884 param_grid = {
57985     'n_estimators': [100, 200, 300],
58086     'max_depth': [10, 20, None],
58187     'min_samples_split': [2, 5, 10],
58288     'min_samples_leaf': [1, 2, 4]
58389 }
58490
58591 # Set up Grid Search
58692 random_search = RandomizedSearchCV(model, param_grid, cv=5, scoring='
587     accuracy', n_jobs=-1, verbose=2)
58893
58994 # Fit on training data
59095 random_search.fit(X_train, y_train)
59196
59297 # Get the best hyperparameters
59398 print("Best Parameters: ", random_search.best_params_)
59499 print("Best Accuracy: %.2f" % random_search.best_score_)
59500
59601 # Update the model with the best parameters
59702 best_model = random_search.best_estimator_
59803
59904 # Fit the best model on the training data
60005 best_model.fit(X_train, y_train)
60106
60207 # Make predictions using the optimized model
60308
60409
60510
60611
60712 ###
60813 #dot_data = tree.export_graphviz(model, out_file=None, feature_names =
609     X_train.columns,class_names = model.classes_,
61014     #         filled=True, rounded=True,
611     leaves_parallel=True, proportion=True)
61215 #graph = graphviz.Source(dot_data)
61316 #graph.render("decision_tree", format="pdf")
61417 X_test = test.drop(columns=['increase_stock'])
61518 y_test = test['increase_stock']
61619 y_predict = best_model.predict(X_test)
61720
61821
61922
62023 print(classification_report(y_test, y_predict))

```

Listing 4: Code for Random forest