
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 *k- Nearest Neighbour* gives best prediction,
8 with accuracy 92%.
9 The group consists of 4 students.

10 **1 Plan**

11 **1.1 From Intro**

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

20 **1.2 From Data analysis task**

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

24 **1.3 From Implementation of methods**

- 25 Each group member should implement one family each, who did what shall be clear!
- 26 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.)
- 27
- 28 Implement a naive version, let's do: *Always low_bike_demand*

29 **1.3.1 What to do with each method**

- 30 1. Implement the method (each person individually)
 - 31 2. Tune hyper-parameters, discuss how this is done (each person individually)
 - 32 3. Evaluate with for example cross-validation. Don't use E_{k-fold} (what is that?) (need to do
 - 33 together)
 - 34 4. (optional) Think about input features, are all relevant? (together)
- 35 Before training, unify pre-processing FOR ALL METHODS and choose ONE OR MULTIPLE
- 36 metrics to evaluate the model. (is it necessary to have the same for all?, is it beneficial?) Examples:
- 37 • accuracy
 - 38 • f1-score
 - 39 • recall
 - 40 • precision

41 Use same test-train split for ALL MODELS

42 **2 Introduction**

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

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

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

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

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

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

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

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

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

52 which algorithm gives the best prediction.

3 Theoretical Background

3.1 Mathematical Overview of the Models

3.1.1 Logistic Regression

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

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

given by the normal equations

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

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

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

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

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

3.1.2 Random forest

The random forest method is based upon decision trees, i.e. dividing the data point into binary groups based on Gini-impurity, entropy or classification error, Gini being the most common. These divisions are then used to create a binary tree shown in figure ??Tree) and where the leaf-nodes are used to classify the target variables based on the input. As of itself the decision tree tends to have unsatisfying results which leads to methods like random forest and sandbagging that boost its accuracy. Sandbagging is a way to sample the data in order to get multiple datasets from the same data. One then creates a decision-tree for every subset data to then combine them into one model. This lessens the variance of the model but increases bias. This means that sandbagging can increase false negatives which in this application makes it nonviable. Random forest on the other hand is viable, it creates multiple trees while discarding random input variable this randomness decreases overfitting creating a more robust model.

3.1.3 Non-parametric method: k-Nearest Neighbour

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

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

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

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

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

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

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%.

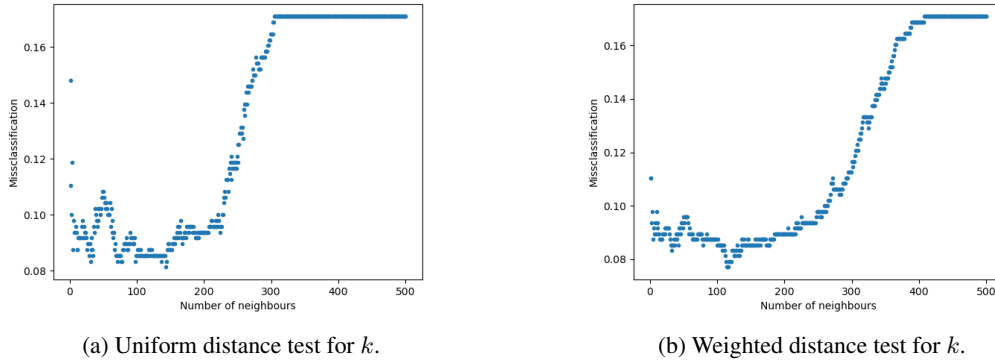


Figure 1: Test for choosing an optimal k -value.

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:

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

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

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

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

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

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

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

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

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

136 3.2 Input Data Modification

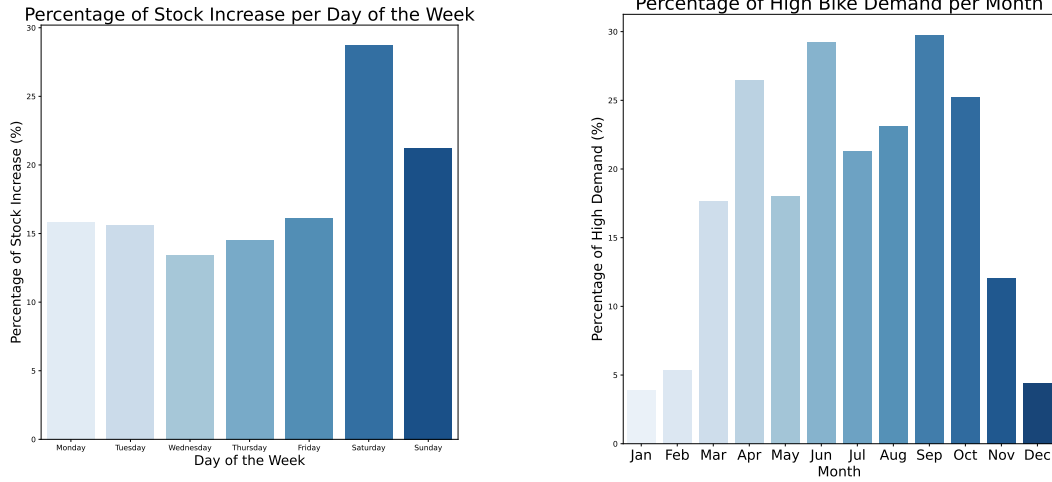
137 By plotting the data and analyzing the .csv file, some observations were made. The different inputs
 138 were then changed accordingly:

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

153 4 Data Analysis

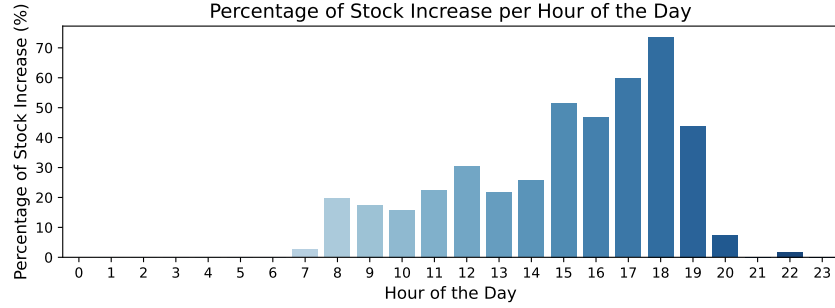
154 In the given data, there are some numerical and categorical features:

- 155 • *Numerical:* temp, dew, humidity, precip, snow, snowdepth, windspeed, cloudcover
- 156 and visibility.
- 157 • *Categorical:* hour_of_day, day_of_week, month, holiday, weekday, summertime, and
- 158 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 where chosen to be the accuracy as well as the precision and recall of the class "high bike demand". The accuracy is defined simply as:

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

And the precision and recall is defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad \text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

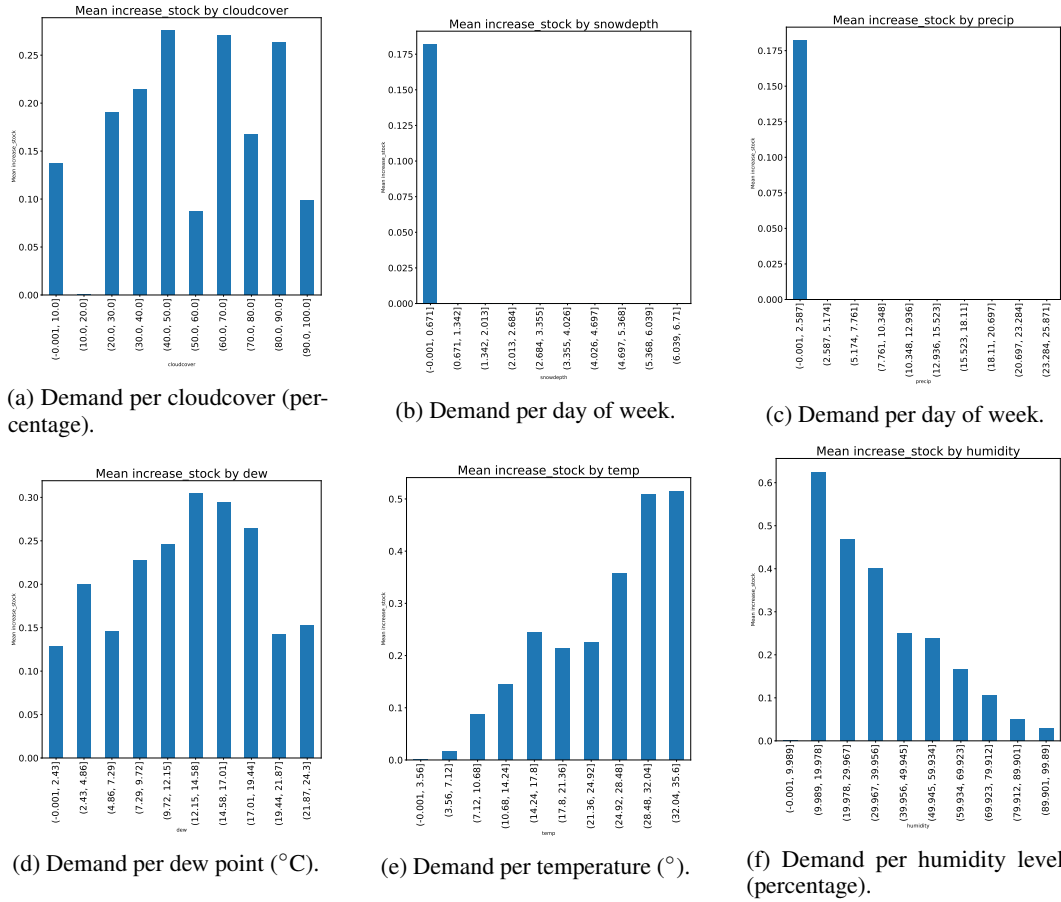


Figure 3: Bike demand vs. various weather parameters.

Furthermore, a naive model that only guessed there is a low demand was compared to the rest of the models. 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

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%
Logistic Regression	90%	73%	63%
Naive	83%	0%	0%

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

6 Conclusion

From the evaluation of the models, k-nn performed the best with the highest accuracy and precision. As for the recall, k-nn did not perform the best but is considered adequate and hence this method is chosen as the best one.

187 One reason for the discriminant analysis falling short of the other models is likely due to
188 these models being designed with the assumption of variables being normally distributed. This is not
189 the case for this particular data set.

190 A Appendix

```

191 1 import pandas as pd
192 2 import numpy as np
193 3 from sklearn.model_selection import train_test_split
194 4 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
195 5 from sklearn.linear_model import LogisticRegression
196 6 from sklearn.metrics import accuracy_score
197 7 from sklearn.metrics import classification_report
198 8
199 9 df = pd.read_csv('training_data_vt2025.csv')
200 10
201 11 # modify the month to represent the periodicity that is observed in
202    data.
203 12 df['month_cos'] = np.cos(df['month']*2*np.pi/12)
204 13 df['month_sin'] = np.sin(df['month']*2*np.pi/12)
205 14
206 15 # time of day, replaced with 3 bool values: is_night, is_day and
207    is_evening,
208 16 # adding the new categories back in the end.
209 17 def categorize_demand(hour):
210 18     if 20 <= hour or 7 >= hour:
211 19         return 'night'
212 20     elif 8 <= hour <= 14:
213 21         return 'day'
214 22     elif 15 <= hour <= 19:
215 23         return 'evening'
216 24
217 25 df['time_of_day'] = df['hour_of_day'].apply(categorize_demand)
218 26 df_dummies = pd.get_dummies(df['time_of_day'], prefix='is', drop_first
219    =False)
220 27 df = pd.concat([df, df_dummies], axis=1)
221 28
222 29 # Create bool of snowdepth and percipitation
223 30 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
224 31 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
225 32
226 33 # Seperate training data from target
227 34 X=df[['#holiday',
228 35     'weekday',
229 36     '#summertime',
230 37     'temp',
231 38     '#dew',
232 39     '#humidity',
233 40     '#visibility',
234 41     '#windspeed',
235 42     '#month',
236 43     'month_cos',
237 44     'month_sin',
238 45     '#hour_of_day',
239 46     'is_day',
240 47     'is_evening',
241 48     'is_night',
242 49     '#hour_cos',
243 50     '#hour_sin',
244 51     'snowdepth_bool',
245 52     'precip_bool'
246 53     ]]
247 54
248 55 y=df['increase_stock']
249 56
250 57 # Split dataset into training and test sets
251 58 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
252    =0.2, random_state=42)
253 59

```

```

25460 # Apply Linear Discriminant Analysis (LDA)
25561 lda = LinearDiscriminantAnalysis(n_components=1)
25662 X_train_lda = lda.fit_transform(X_train, y_train)
25763 X_test_lda = lda.transform(X_test)
25864
25965 # Train a classifier (Logistic Regression)
26066 clf = LogisticRegression()
26167 clf.fit(X_train_lda, y_train)
26268
26369 # Make predictions
26470 y_pred = clf.predict(X_test_lda)
26571
26672 # Evaluate accuracy
26773 accuracy = accuracy_score(y_test, y_pred)
26874 print(f"Model Accuracy: {accuracy:.2f}")
26975
27076 print(classification_report(y_test, y_pred))

```

Listing 1: Code for LDA

```

271 1 import pandas as pd
272 2 import numpy as np
273 3 from sklearn.model_selection import train_test_split
274 4 from sklearn.discriminant_analysis import
275     QuadraticDiscriminantAnalysis
276 5 from sklearn.metrics import accuracy_score
277 6 from sklearn.metrics import classification_report
278 7
279 8 df = pd.read_csv('training_data_vt2025.csv')
280 9
28110 # modify the month to represent the periodicity that is observed in
282     data.
28311 df['month_cos'] = np.cos(df['month']*2*np.pi/12)
28412 df['month_sin'] = np.sin(df['month']*2*np.pi/12)
28513
28614 # time of day, replaced with 3 bool values: is_night, is_day and
287     is_evening,
28815 # adding the new categories back in the end.
28916 def categorize_demand(hour):
29017     if 20 <= hour or 7 >= hour:
29118         return 'night'
29219     elif 8 <= hour <= 14:
29320         return 'day'
29421     elif 15 <= hour <= 19:
29522         return 'evening'
29623
29724 df['time_of_day'] = df['hour_of_day'].apply(categorize_demand)
29825 df_dummies = pd.get_dummies(df['time_of_day'], prefix='is', drop_first
299     =False)
30026 df = pd.concat([df, df_dummies], axis=1)
30127
30228 # Create bool of snowdepth and percipitation
30329 df['snowdepth_bool'] = df['snowdepth'].where(df['snowdepth'] == 0, 1)
30430 df['precip_bool'] = df['precip'].where(df['precip'] == 0, 1)
30531
30632 # Seperate training data from target
30733 X=df[['#holiday',
30834     'weekday',
30935     '#summertime',
31036     'temp',
31137     '#dew',
31238     '#humidity',
31339     '#visibility',
31440     '#windspeed',
31541     '#month',

```

```

31612         'month_cos',
31743         'month_sin',
31844         #'hour_of_day',
31945         'is_day',
32046         'is_evening',
32147         'is_night',
32248         #'snowdepth_bool',
32349         'precip_bool'
32450     ]]
32551
32652 y=df['increase_stock']
32753
32854 # Split dataset into training and test sets
32955 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
330         =0.2, random_state=42)
33156
33257 # Apply Quadratic Discriminant Analysis (QDA)
33358 qda = QuadraticDiscriminantAnalysis()
33459 X_train_lda = qda.fit(X_train, y_train)
33560
33661 # Make predictions
33762 y_pred = qda.predict(X_test)
33863
33964 # Evaluate accuracy
34065 accuracy = accuracy_score(y_test, y_pred)
34166 print(f"Model Accuracy: {accuracy:.2f}")
34267
34368 print(classification_report(y_test, y_pred))

```

Listing 2: Code for QDA

```

344 1 import pandas as pd
345 2 import numpy as np
346 3 import matplotlib
347 4 import matplotlib.pyplot as plt
348 5 from sklearn import tree
349 6 from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
350 7 import graphviz
351 8 from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
352 9 from sklearn.metrics import classification_report
35310
35411 df = pd.read_csv('training_data_vt2025.csv')
35512 #df.info()
35613
35714 # Modify the dataset, emphasizing different variables
35815 df.iloc[:,12]=df.iloc[:,12]**2
35916 df.iloc[:,13]=np.sqrt(df.iloc[:,13])
36017 df.iloc[:,11] = df.iloc[:,11]**2
36118
36219 df['month_cos'] = np.cos(df.month*np.pi/12)
36320 df['month_sin'] = np.sin(df.month*np.pi/12)
36421
36522 # time of day, replaed with low,medium and high demand,
36623 # adding the new categories back in the end.
36724 def categorize_demand(hour):
36825     if 20 <= hour or 7 >= hour:
36926         return 'night'
37027     elif 8 <= hour <= 14:
37128         return 'day'
37229     elif 15 <= hour <= 19:
37330         return 'evening'
37431
37532 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
37633 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
377         drop_first=False)

```

```

37834 df = pd.concat([df, df_dummies], axis=1)
37935
38036 # converting to bools
38137 def if_zero(data):
38238     if data == 0:
38339         return True
38440     else:
38541         return False
38642
38743 # temperature
38844
38945 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
39046 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
39147
39248 # Split into train and test:
39349
39450 #df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
39551 #df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
39652 np.random.seed(0)
39753
39854 df_modified=df[['holiday',
39955                 'weekday',
40056                 'summertime',
40157                 'temp',
40258                 'dew',
40359                 'humidity',
40460                 'visibility',
40561                 'windspeed',
40662                 'month_cos',
40763                 'month_sin',
40864                 'demand_day',
40965                 'demand_evening',
41066                 'demand_night',
41167                 'snowdepth_bool',
41268                 'precip_bool',
41369                 'increase_stock']]
41470
41571 N = df_modified.shape[0]
41672 n = round(0.7*N)
41773 trainI = np.random.choice(N,size=n,replace=False)
41874 trainIndex = df_modified.index.isin(trainI)
41975 train = df_modified.iloc[trainIndex]
42076 test = df_modified.iloc[~trainIndex]
42177
42278 X_train = train.drop(columns=['increase_stock'])
42379 # Need to transform the qualitative variables to dummy variables
42480
42581 y_train = train['increase_stock']
42682
42783 model = RandomForestClassifier(random_state=42)
42884 param_grid = {
42985     'n_estimators': [100, 200, 300],
43086     'max_depth': [10, 20, None],
43187     'min_samples_split': [2, 5, 10],
43288     'min_samples_leaf': [1, 2, 4]
43389 }
43490
43591 # Set up Grid Search
43692 random_search = RandomizedSearchCV(model, param_grid, cv=5, scoring='
437     accuracy', n_jobs=-1, verbose=2)
43893
43994 # Fit on training data
44095 random_search.fit(X_train, y_train)
44196
44297 # Get the best hyperparameters

```

```

4438 print("Best Parameters: ", random_search.best_params_)
4449 print("Best Accuracy: %.2f" % random_search.best_score_)
4450
4461 # Update the model with the best parameters
4472 best_model = random_search.best_estimator_
4483
4494 # Fit the best model on the training data
4505 best_model.fit(X_train, y_train)
4516
4527 # Make predictions using the optimized model
4538
4549
4550
4561
4572 ###
4583 #dot_data = tree.export_graphviz(model, out_file=None, feature_names =
459     X_train.columns, class_names = model.classes_,
4604 #                                filled=True, rounded=True,
461     leaves_parallel=True, proportion=True)
4625 #graph = graphviz.Source(dot_data)
4636 #graph.render("decision_tree", format="pdf")
4647 X_test = test.drop(columns=['increase_stock'])
4658 y_test = test['increase_stock']
4669 y_predict = best_model.predict(X_test)
4670
4681
4692
4703 print(classification_report(y_test, y_predict))

```

Listing 3: Code for Random Forest

```

4711 import numpy as np
4722 import pandas as pd
4733 import matplotlib.pyplot as plt
4744 import sklearn.linear_model as skl_lm
4755 import sklearn.preprocessing as pp
4766 import sklearn.metrics as skl_m
4777
4788 import sklearn.neighbors as skl_nb
4799
4800 df = pd.read_csv('training_data_vt2025.csv')
4811 #df.info()
4822
4833 # Modify the dataset, emphasizing different variables
4844 #df.iloc[:,12]=df.iloc[:,12]**2
4855 #df.iloc[:,13]=np.sqrt(df.iloc[:,13])
4866 #df.iloc[:,11] = df.iloc[:,11]**2
4877
4888 df['month_cos'] = np.cos(df.month*np.pi/12)
4899 df['month_sin'] = np.sin(df.month*np.pi/12)
4900
49121 # time of day, replaed with low,medium and high demand,
49222 # adding the new categories back in the end.
49323 def categorize_demand(hour):
49424     if 20 <= hour or 7 >= hour:
49525         return 'night'
49626     elif 8 <= hour <= 14:
49727         return 'day'
49828     elif 15 <= hour <= 19:
49929         return 'evening'
5000
50131 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
50232 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
503     drop_first=False)
50433 df = pd.concat([df, df_dummies], axis=1)

```

```

50534
50635 # converting to bools
50736 def if_zero(data):
50837     if data == 0:
50938         return True
51039     else:
51140         return False
51241
51342 # temperature
51443
51544 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
51645 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
51746
51847 # Split into train and test:
51948
52049 #df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
52150 #df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
52251 np.random.seed(0)
52352
52453 df_modified=df[['holiday',
52554                 'weekday',
52655                 '#summertime',
52756                 'temp',
52857                 '#dew',
52958                 'humidity',
53059                 'visibility',
53160                 'windspeed',
53261                 'month_cos',
53362                 'month_sin',
53463                 'demand_day',
53564                 'demand_evening',
53665                 'demand_night',
53766                 'snowdepth_bool',
53867                 'precip_bool',
53968                 'increase_stock']]
54069
54170 N = df_modified.shape[0]
54271 n = round(0.7*N)
54372 trainI = np.random.choice(N,size=n,replace=False)
54473 trainIndex = df_modified.index.isin(trainI)
54574 train = df_modified.iloc[trainIndex]
54675 test = df_modified.iloc[~trainIndex]
54776
54877 # Set up X,Y
54978
55079 # Train data
55180 X = train.iloc[:,0:-2]
55281 Y = train['increase_stock']
55382
55483 # Test data
55584 X_test = test.iloc[:,0:-2]
55685 Y_test = test['increase_stock']
55786
55887
55988 """
56089 # Tests for k-value
56190 # TEST 1 - uniform distance
56291 missclassification = []
56392 for k in range(500): # Try n_neighbours = 1, 2, ...,
56493
56594     #kNN method
56695     scaler = pp.StandardScaler().fit(X)
56796     model = skl_nb.KNeighborsClassifier(n_neighbors = k+1, weights = '
56897     uniform')
56997     model.fit(scaler.transform(X),Y)

```

```

5708
5719 # Prediction
5720 y_hat = model.predict(scaler.transform(X_test))
5731 missclassification.append(np.mean(y_hat != Y_test))
5742
5753 K = np.linspace(1, 500, 500)
5764 plt.plot(K, missclassification, '.')
5775 plt.ylabel('Missclassification')
5786 plt.xlabel('Number of neighbours')
5797 plt.show()
5808
58109 #TEST 2
5820 missclassification = []
5831 for k in range(500): # Try n_neighbours = 1, 2, ...,
5842
5853     #kNN method
5864     scaler = pp.StandardScaler().fit(X)
5875     model = skl_nb.KNeighborsClassifier(n_neighbors = k+1, weights = '
588     distance')
5896     model.fit(scaler.transform(X),Y)
5907
5918     # Prediction
5929     y_hat = model.predict(scaler.transform(X_test))
5930     missclassification.append(np.mean(y_hat != Y_test))
5941
5952 K = np.linspace(1, 500, 500)
5963 plt.plot(K, missclassification, '.')
5974 plt.ylabel('Missclassification')
5985 plt.xlabel('Number of neighbours')
5996 plt.show()
6007 """
6018
6029
6030
60431 # creating the model
6052 model = skl_nb.KNeighborsClassifier(n_neighbors = 120, weights = '
606     distance')
60733
6084
6095 # Scaling the data, otherwise
6106 scaler = pp.StandardScaler().fit(X)
6117 model.fit(scaler.transform(X),Y)
6128 y_hat = model.predict(scaler.transform(X_test))
6139
6140
6151
6162 '''
6173 # oskalad data
6184 model.fit(X,Y)
6195 y_hat = model.predict(X_test)'''
6206
6217 # Get confusion matrix
6228 diff = pd.crosstab(y_hat, Y_test)
6239 print(f'Confusion matrix: \n {diff}')
6240
6251 # No. of TP,TN,FP,FN
6262 '''TP = diff.iloc[0,0]
6273 TN = diff.iloc[1,1]
6284 FP = diff.iloc[1,0]
6295 FN = diff.iloc[0,1]'''
6306
6317 # Get metrics:
6328 print(skl_m.classification_report(Y_test, y_hat))

```

Listing 4: Code for K- nearest neighbours

```

633 1 import numpy as np
634 2 import pandas as pd
635 3 import matplotlib.pyplot as plt
636 4 import sklearn.linear_model as skl_lm
637 5 import sklearn.preprocessing as pp
638 6 import sklearn.metrics as skl_m
639 7
640 8 df = pd.read_csv('training_data_vt2025.csv')
641 9 #df.info()
642 10
643 11 # Modify the dataset, emphasizing different variables
644 12 #df.iloc[:,12]=df.iloc[:,12]**2
645 13 #df.iloc[:,13]=np.sqrt(df.iloc[:,13])
646 14 #df.iloc[:,11] = df.iloc[:,11]**2
647 15
648 16 df['month_cos'] = np.cos(df.month*np.pi/12)
649 17 df['month_sin'] = np.sin(df.month*np.pi/12)
650 18
651 19 # time of day, replaed with low,medium and high demand,
652 20 # adding the new categories back in the end.
653 21 def categorize_demand(hour):
654 22     if 20 <= hour or 7 >= hour:
655 23         return 'night'
656 24     elif 8 <= hour <= 14:
657 25         return 'day'
658 26     elif 15 <= hour <= 19:
659 27         return 'evening'
660 28
661 29 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
662 30 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
663 31 drop_first=False)
664 32 df = pd.concat([df, df_dummies], axis=1)
665 33
666 34 # converting to bools
667 35 def if_zero(data):
668 36     if data == 0:
669 37         return True
670 38     else:
671 39         return False
672 40
673 41 # temperature
674 42
675 43 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
676 44 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
677 45
678 46 # Split into train and test:
679 47
680 48 #df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
681 49 #df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
682 50 np.random.seed(0)
683 51
684 52 df_modified=df[['holiday',
685 53                'weekday',
686 54                'summertime',
687 55                'temp',
688 56                'dew',
689 57                'humidity',
690 58                'visibility',
691 59                'windspeed',
692 60                'month_cos',
693 61                'month_sin',
694 62                'demand_day',
695 63                'demand_evening',
696 64                'demand_night',
697 65                'snowdepth_bool',

```



```

69855         'precip_bool',
69956         'increase_stock'']]
70057
70168 N = df_modified.shape[0]
70259 n = round(0.7*N)
70370 trainI = np.random.choice(N,size=n,replace=False)
70471 trainIndex = df_modified.index.isin(trainI)
70572 train = df_modified.iloc[trainIndex]
70673 test = df_modified.iloc[~trainIndex]
70774
70875 # Set up X,Y
70976
71077 # Train data
71178 X = train.iloc[:,0:-2]
71279 Y = train['increase_stock']
71380
71481 # Test data
71582 X_test = test.iloc[:,0:-2]
71683 Y_test = test['increase_stock']
71784
71885 model = skl_lm.LogisticRegression()
71986
72087 # Scaling the data, otherwise
72188 scaler = pp.StandardScaler().fit(X)
72289 model.fit(scaler.transform(X),Y)
72390 y_hat = model.predict(scaler.transform(X_test))
72491
72592 '''
72693 # oskalad data
72794 model.fit(X,Y)
72895 y_hat = model.predict(X_test)'''
72996
73097 # Get confusion matrix
73198 diff = pd.crosstab(y_hat, Y_test)
73299 print(f'Confusion matrix: \n {diff}')
73300
73401 # No. of TP,TN,FP,FN
73502 '''TP = diff.iloc[0,0]
73603 TN = diff.iloc[1,1]
73704 FP = diff.iloc[1,0]
73805 FN = diff.iloc[0,1]'''
73906
74007 # Get metrics:
74108 print(skl_m.classification_report(Y_test, y_hat))

```

Listing 5: Code for Logistic Regression