

---

# 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.

# 1 Introduction

Statistical machine learning is a subject that aims to build and train algorithms, that analyse large amount of data, and make predictions for the future, which are computed by using established statistical models, and tools from functional analysis. This is a project in supervised, statistical machine learning, where several models were created, and trained, in order to analyse which one of them gives best prediction for the project "Do we need more bikes", where we want to understand, and predict if there is a high, or low demand of city bikes in the public transportation of Washington, a project by the District Department of Transportation in Washington D.C..

The data set used for training our models, consist of 15 variables, containing quantitative/qualitative data. We developed several models, and evaluated them with cross-validation, in order to understand which algorithm gives the best prediction.

## 2 Theoretical Background

### 2.1 Mathematical Overview of the Models

#### 2.1.1 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,  $\theta$  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.

#### 2.1.2 Random forest

The random forest method is a 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 thee leaf-nodes are used to classify the target variables bases on the input. As of itself the disition tree tends to have unsatisfying results which leads to methodes like random forest and sandbagging that boost its accuracy. Sandbagging is a way to sampel the data in order to get multiple datasets from the same data. One then creates a desition-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 theis aplication makes i nonviable. Random forest on the otherhand is viable, it creates mutple trees whilse disrecarding random input variable this randomnes decreases overfitting creating a more robust model.

#### 2.1.3 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 alias *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 belong to the high demand-class. Denote  $c = \text{high demand class}$ . For simplicity, assume Euclidean ambience. 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,  $\chi$  is the characteristic function

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

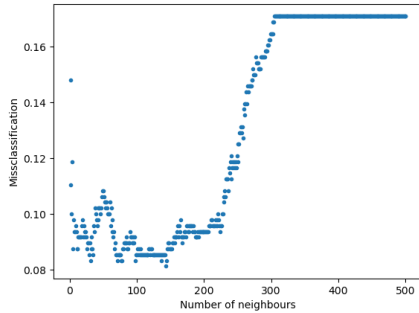
It is very common to use a weighted sum to predict the next value, i.e.

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

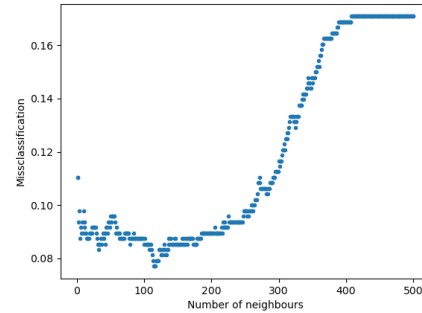
where  $d$  is the standard Euclidean metric, computing the distance between an input  $x$ , and a neighbour  $x_n$ .

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%.



(a) Uniform distance test for  $k$ .



(b) Weighted distance test for  $k$ .

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

#### 2.1.4 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:

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

81 For the discrete version it is obtained:

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

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

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

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

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

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

89 Where  $\mathbf{x}$  is the  $d$ -dimensional data point,  $\mu$  is the ( $d$ -dimensional) mean of the random variable.  $\Sigma$  is  
90 the symmetric, positive definite covariance matrix defined by:

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

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

93  
94 Quadratic discriminant analysis (QDA) is heavily based on LDA with the sole difference  
95 being how the covariance matrix  $\Sigma$  is created. In LDA, the covariance matrix is assumed to be the  
96 same for data in each and every class. In QDA however, the covariance matrix is calculated for each  
97 class as follows:

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

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

## 104 2.2 Input Data Modification

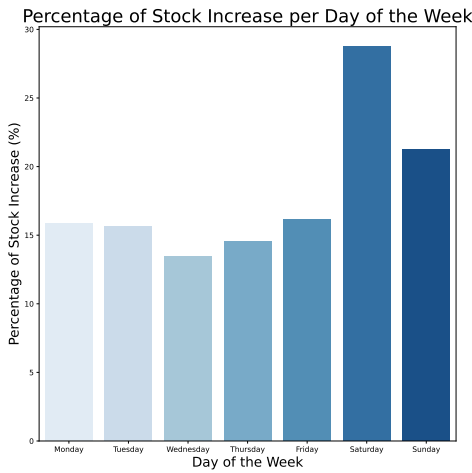
105 By plotting the data and analyzing the .csv file, some observations were made. The different inputs  
106 were then changed accordingly:

- 107 • *Kept as-is*: weekday, windspeed, visibility, temp
- 108 • *Modified*:
  - 109 – month - split into two inputs, one cosine and one sine part. This makes the new inputs
  - 110 linear and can follow the fluctuations of the year. The original input was discarded.
  - 111 – hour\_of\_day - split into three boolean variables: demand\_day, demand\_evening,
  - 112 and demand\_night, reflecting if the time was between 08-14, 15-19 or 20-07 respec-
  - 113 tively. This was done because plotting the data showed three different plateaus of
  - 114 demand for the different time intervals. The original input was discarded.
  - 115 – snowdepth, precip were transformed into booleans, reflecting if it was raining or
  - 116 if there was snow on the ground or not. This was done as there was no times where
  - 117 demand was high when it was raining or when there was snow on the ground.
- 118 • *Removed*: cloudcover, day\_of\_week, snow, dew, holiday, summertime. These were
- 119 removed due to being redundant (e.g. summertime), not showing a clear trend (e.g.
- 120 cloudcover), giving a worse score when used, or all three (e.g. day\_of\_week).

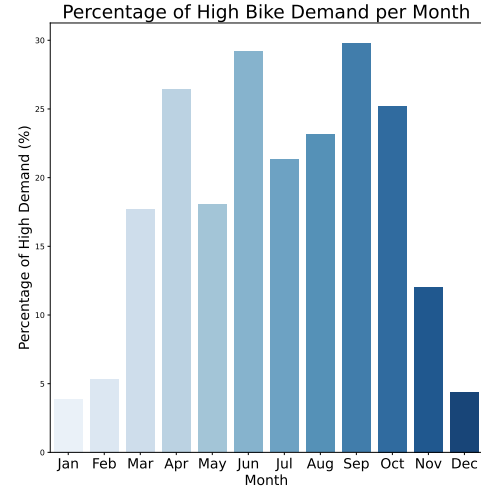
### 3 Data Analysis

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

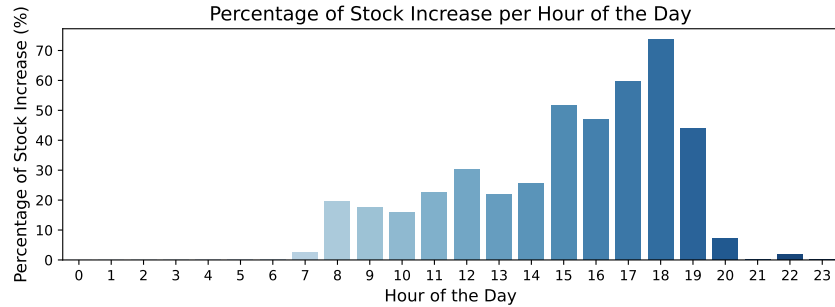
- *Numerical*: temp, dew, humidity, precip, snow, snowdepth, windspeed, cloudcover and visibility.
- *Categorical*: hour\_of\_day, day\_of\_week, month, holiday, weekday, summertime, and increase\_stock



(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 CO<sub>2</sub> emissions.

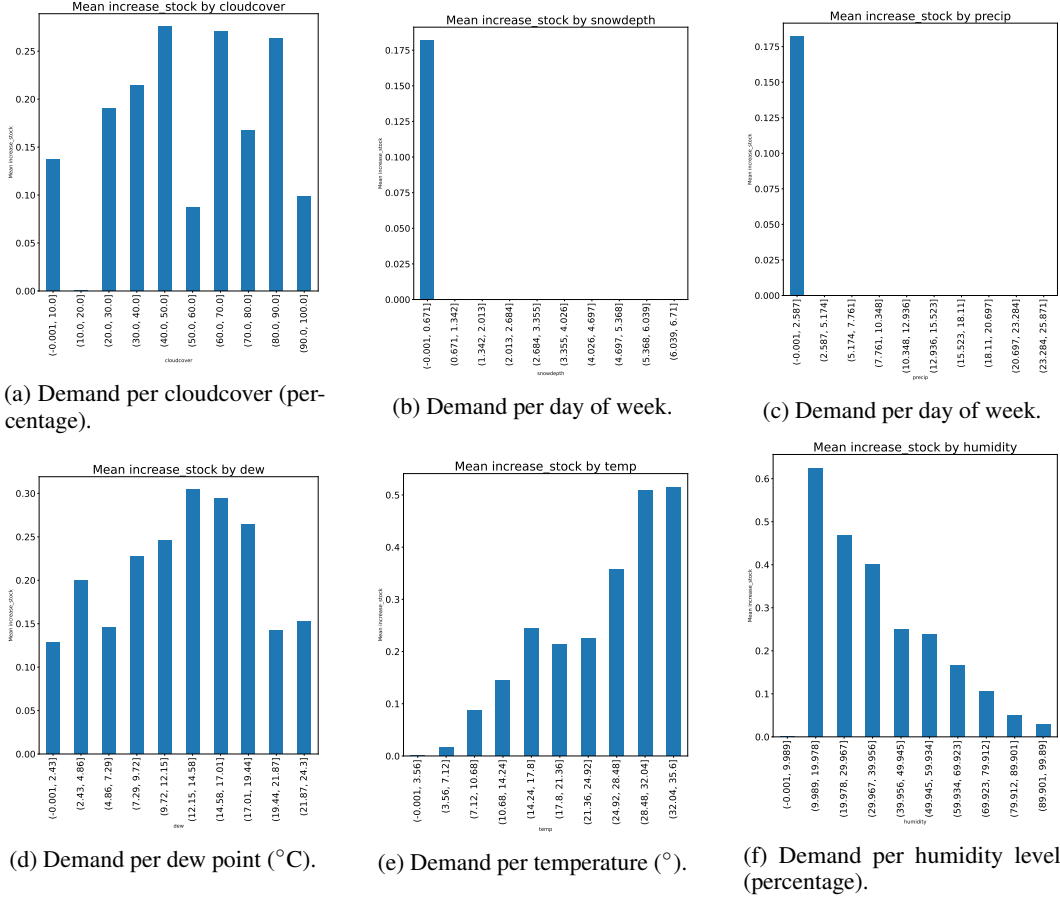


Figure 3: Bike demand vs. various weather parameters.

## 4 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:

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

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%

146 outperforming linear and quadratic regression on accuracy, precision and recall. Out of random forest  
147 and kNN the group would proceed with the kNN method, its higher accuracy and precision score out  
148 waying the slightly better recall score of random forest. This will mean a slight loss in income caused  
149 by increasing false negatives but is thought to be covered by fewer false positives.

## 150 **5 Conclusion**

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

154  
155 One reason for the discriminant analysis falling short of the other models is likely due to  
156 these models being designed with the assumption of variables being normally distributed. This is not  
157 the case for this particular data set.

## 158 A Appendix

```

159 1 import pandas as pd
160 2 import numpy as np
161 3 from sklearn.model_selection import train_test_split
162 4 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
163 5 from sklearn.linear_model import LogisticRegression
164 6 from sklearn.metrics import accuracy_score
165 7 from sklearn.metrics import classification_report
166 8
167 9 df = pd.read_csv('training_data_vt2025.csv')
168 10
169 11 # modify the month to represent the periodicity that is observed in
170 12 data.
171 13 df['month_cos'] = np.cos(df['month']*2*np.pi/12)
172 14 df['month_sin'] = np.sin(df['month']*2*np.pi/12)
173 15
174 16 # time of day, replaced with 3 bool values: is_night, is_day and
175 17 is_evening,
176 18 # adding the new categories back in the end.
177 19 def categorize_demand(hour):
178 20     if 20 <= hour or 7 >= hour:
179 21         return 'night'
180 22     elif 8 <= hour <= 14:
181 23         return 'day'
182 24     elif 15 <= hour <= 19:
183 25         return 'evening'
184 26
185 27 df['time_of_day'] = df['hour_of_day'].apply(categorize_demand)
186 28 df_dummies = pd.get_dummies(df['time_of_day'], prefix='is', drop_first
187 29 =False)
188 30 df = pd.concat([df, df_dummies], axis=1)
189 31
190 32 # Create bool of snowdepth and percipitation
191 33 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
192 34 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
193 35
194 36 # Seperate training data from target
195 37 X=df[['#holiday',
196 38     'weekday',
197 39     '#summertime',
198 40     'temp',
199 41     '#dew',
200 42     '#humidity',
201 43     '#visibility',
202 44     '#windspeed',
203 45     '#month',
204 46     'month_cos',
205 47     'month_sin',
206 48     '#hour_of_day',
207 49     'is_day',
208 50     'is_evening',
209 51     'is_night',
210 52     '#hour_cos',
211 53     '#hour_sin',
212 54     'snowdepth_bool',
213 55     'precip_bool'
214 56 ]]
215 57
216 58 y=df['increase_stock']
217 59
218 60 # Split dataset into training and test sets
219 61 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
220 62 =0.2, random_state=42)
221 63

```



```

22250 # Apply Linear Discriminant Analysis (LDA)
22351 lda = LinearDiscriminantAnalysis(n_components=1)
22452 X_train_lda = lda.fit_transform(X_train, y_train)
22553 X_test_lda = lda.transform(X_test)
22654
22755 # Train a classifier (Logistic Regression)
22856 clf = LogisticRegression()
22957 clf.fit(X_train_lda, y_train)
23058
23159 # Make predictions
23260 y_pred = clf.predict(X_test_lda)
23361
23462 # Evaluate accuracy
23563 accuracy = accuracy_score(y_test, y_pred)
23664 print(f"Model Accuracy: {accuracy:.2f}")
23765
23866 print(classification_report(y_test, y_pred))

```

Listing 1: Code for LDA

```

239 1 import pandas as pd
240 2 import numpy as np
241 3 from sklearn.model_selection import train_test_split
242 4 from sklearn.discriminant_analysis import
243     QuadraticDiscriminantAnalysis
244 5 from sklearn.metrics import accuracy_score
245 6 from sklearn.metrics import classification_report
246 7
247 8 df = pd.read_csv('training_data_vt2025.csv')
248 9
24910 # modify the month to represent the periodicity that is observed in
250     data.
25111 df['month_cos'] = np.cos(df['month']*2*np.pi/12)
25212 df['month_sin'] = np.sin(df['month']*2*np.pi/12)
25313
25414 # time of day, replaced with 3 bool values: is_night, is_day and
255     is_evening,
25615 # adding the new categories back in the end.
25716 def categorize_demand(hour):
25817     if 20 <= hour or 7 >= hour:
25918         return 'night'
26019     elif 8 <= hour <= 14:
26120         return 'day'
26221     elif 15 <= hour <= 19:
26322         return 'evening'
26423
26524 df['time_of_day'] = df['hour_of_day'].apply(categorize_demand)
26625 df_dummies = pd.get_dummies(df['time_of_day'], prefix='is', drop_first
267     =False)
26826 df = pd.concat([df, df_dummies], axis=1)
26927
27028 # Create bool of snowdepth and percipitation
27129 df['snowdepth_bool'] = df['snowdepth'].where(df['snowdepth'] == 0, 1)
27230 df['precip_bool'] = df['precip'].where(df['precip'] == 0, 1)
27331
27432 # Seperate training data from target
27533 X=df[['#holiday',
27634     'weekday',
27735     '#summertime',
27836     'temp',
27937     '#dew',
28038     '#humidity',
28139     '#visibility',
28240     '#windspeed',
28341     '#month',

```

```

28412         'month_cos',
28513         'month_sin',
28614         #'hour_of_day',
28715         'is_day',
28816         'is_evening',
28917         'is_night',
29018         #'snowdepth_bool',
29119         'precip_bool'
29220     ]]
29321
29422 y=df['increase_stock']
29523
29624 # Split dataset into training and test sets
29725 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
298         =0.2, random_state=42)
29926
30027 # Apply Quadratic Discriminant Analysis (QDA)
30128 qda = QuadraticDiscriminantAnalysis()
30229 X_train_lda = qda.fit(X_train, y_train)
30330
30431 # Make predictions
30532 y_pred = qda.predict(X_test)
30633
30734 # Evaluate accuracy
30835 accuracy = accuracy_score(y_test, y_pred)
30936 print(f"Model Accuracy: {accuracy:.2f}")
31037
31138 print(classification_report(y_test, y_pred))

```

Listing 2: Code for QDA

```

3121 import pandas as pd
3132 import numpy as np
3143 import matplotlib
3154 import matplotlib.pyplot as plt
3165 from sklearn import tree
3176 from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
3187 import graphviz
3198 from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
3209 from sklearn.metrics import classification_report
3210
32211 df = pd.read_csv('training_data_vt2025.csv')
32312 #df.info()
32413
32514 # Modify the dataset, emphasizing different variables
32615 df.iloc[:,12]=df.iloc[:,12]**2
32716 df.iloc[:,13]=np.sqrt(df.iloc[:,13])
32817 df.iloc[:,11] = df.iloc[:,11]**2
32918
33019 df['month_cos'] = np.cos(df.month*np.pi/12)
33120 df['month_sin'] = np.sin(df.month*np.pi/12)
33221
33322 # time of day, replaed with low,medium and high demand,
33423 # adding the new categories back in the end.
33524 def categorize_demand(hour):
33625     if 20 <= hour or 7 >= hour:
33726         return 'night'
33827     elif 8 <= hour <= 14:
33928         return 'day'
34029     elif 15 <= hour <= 19:
34130         return 'evening'
34231
34332 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
34433 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
345         drop_first=False)

```

```

34634 df = pd.concat([df, df_dummies], axis=1)
34735
34836 # converting to bools
34937 def if_zero(data):
35038     if data == 0:
35139         return True
35240     else:
35341         return False
35442
35543 # temperature
35644
35745 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
35846 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
35947
36048 # Split into train and test:
36149
36250 df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
36351 df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
36452 np.random.seed(0)
36553
36654 df_modified=df[ ['#holiday',
36755                  'weekday',
36856                  '#summertime',
36957                  'temp',
37058                  '#dew',
37159                  '#humidity',
37260                  'visibility',
37361                  'windspeed',
37462                  'month_cos',
37563                  'month_sin',
37664                  'demand_day',
37765                  'demand_evening',
37866                  'demand_night',
37967                  'snowdepth_bool',
38068                  'precip_bool',
38169                  'increase_stock']]
38270
38371 N = df_modified.shape[0]
38472 n = round(0.7*N)
38573 trainI = np.random.choice(N,size=n,replace=False)
38674 trainIndex = df_modified.index.isin(trainI)
38775 train = df_modified.iloc[trainIndex]
38876 test = df_modified.iloc[~trainIndex]
38977
39078 X_train = train.drop(columns=['increase_stock'])
39179 # Need to transform the qualitative variables to dummy variables
39280
39381 y_train = train['increase_stock']
39482
39583 model = RandomForestClassifier(random_state=42)
39684 param_grid = {
39785     'n_estimators': [100, 200, 300],
39886     'max_depth': [10, 20, None],
39987     'min_samples_split': [2, 5, 10],
40088     'min_samples_leaf': [1, 2, 4]
40189 }
40290
40391 # Set up Grid Search
40492 random_search = RandomizedSearchCV(model, param_grid, cv=5, scoring='
40593     accuracy', n_jobs=-1, verbose=2)
40694
40794 # Fit on training data
40895 random_search.fit(X_train, y_train)
40996
41097 # Get the best hyperparameters

```

```

41198 print("Best Parameters: ", random_search.best_params_)
41299 print("Best Accuracy: %.2f" % random_search.best_score_)
41300
41401 # Update the model with the best parameters
41502 best_model = random_search.best_estimator_
41603
41704 # Fit the best model on the training data
41805 best_model.fit(X_train, y_train)
41906
42007 # Make predictions using the optimized model
42108
42209
42300
42401
42502 ###
42603 #dot_data = tree.export_graphviz(model, out_file=None, feature_names =
427     X_train.columns, class_names = model.classes_,
42804 #                                filled=True, rounded=True,
429     leaves_parallel=True, proportion=True)
43005 #graph = graphviz.Source(dot_data)
43106 #graph.render("decision_tree", format="pdf")
43207 X_test = test.drop(columns=['increase_stock'])
43308 y_test = test['increase_stock']
43409 y_predict = best_model.predict(X_test)
43500
43601
43702
43803 print(classification_report(y_test, y_predict))

```

Listing 3: Code for Random Forest

```

439 1 import numpy as np
440 2 import pandas as pd
441 3 import matplotlib.pyplot as plt
442 4 import sklearn.linear_model as skl_lm
443 5 import sklearn.preprocessing as pp
444 6 import sklearn.metrics as skl_m
445 7
446 8 import sklearn.neighbors as skl_nb
447 9
4480 df = pd.read_csv('training_data_vt2025.csv')
44901 #df.info()
45002
45103 # Modify the dataset, emphasizing different variables
45204 #df.iloc[:,12]=df.iloc[:,12]**2
45305 #df.iloc[:,13]=np.sqrt(df.iloc[:,13])
45406 #df.iloc[:,11] = df.iloc[:,11]**2
45507
45608 df['month_cos'] = np.cos(df.month*np.pi/12)
45709 df['month_sin'] = np.sin(df.month*np.pi/12)
45800
45901 # time of day, replaed with low,medium and high demand,
46002 # adding the new categories back in the end.
46103 def categorize_demand(hour):
46204     if 20 <= hour or 7 >= hour:
46305         return 'night'
46406     elif 8 <= hour <= 14:
46507         return 'day'
46608     elif 15 <= hour <= 19:
46709         return 'evening'
46800
46901 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
47002 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
471     drop_first=False)
47203 df = pd.concat([df, df_dummies], axis=1)

```

```

47334
47335 # converting to bools
47336 def if_zero(data):
47337     if data == 0:
47338         return True
47339     else:
47340         return False
47341
47342 # temperature
47343
47344 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
47345 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
47346
47347 # Split into train and test:
47348
47349 #df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
47350 #df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
47351 np.random.seed(0)
47352
47353 df_modified=df[['holiday',
47354                 'weekday',
47355                 '#summertime',
47356                 'temp',
47357                 '#dew',
47358                 'humidity',
47359                 'visibility',
47360                 'windspeed',
47361                 'month_cos',
47362                 'month_sin',
47363                 'demand_day',
47364                 'demand_evening',
47365                 'demand_night',
47366                 'snowdepth_bool',
47367                 'precip_bool',
47368                 'increase_stock']]
47369
47370 N = df_modified.shape[0]
47371 n = round(0.7*N)
47372 trainI = np.random.choice(N,size=n,replace=False)
47373 trainIndex = df_modified.index.isin(trainI)
47374 train = df_modified.iloc[trainIndex]
47375 test = df_modified.iloc[~trainIndex]
47376
47377 # Set up X,Y
47378
47379 # Train data
47380 X = train.iloc[:,0:-2]
47381 Y = train['increase_stock']
47382
47383 # Test data
47384 X_test = test.iloc[:,0:-2]
47385 Y_test = test['increase_stock']
47386
47387
47388 """
47389 # Tests for k-value
47390 # TEST 1 - uniform distance
47391 missclassification = []
47392 for k in range(500): # Try n_neighbours = 1, 2, ...,
47393
47394     #kNN method
47395     scaler = pp.StandardScaler().fit(X)
47396     model = skl_nb.KNeighborsClassifier(n_neighbors = k+1, weights = '
47397     uniform')
47398     model.fit(scaler.transform(X),Y)

```

```

53898
53999     # Prediction
54000     y_hat = model.predict(scaler.transform(X_test))
54101     missclassification.append(np.mean(y_hat != Y_test))
54202
54303 K = np.linspace(1, 500, 500)
54404 plt.plot(K, missclassification, '.')
54505 plt.ylabel('Missclassification')
54606 plt.xlabel('Number of neighbours')
54707 plt.show()
54808
54909 #TEST 2
55010 missclassification = []
55111 for k in range(500): # Try n_neighbours = 1, 2, ...,
55212
55313     #kNN method
55414     scaler = pp.StandardScaler().fit(X)
55515     model = skl_nb.KNeighborsClassifier(n_neighbors = k+1, weights = '
55616     distance')
55717     model.fit(scaler.transform(X),Y)
55818
55919     # Prediction
56020     y_hat = model.predict(scaler.transform(X_test))
56121     missclassification.append(np.mean(y_hat != Y_test))
56222
56323 K = np.linspace(1, 500, 500)
56424 plt.plot(K, missclassification, '.')
56525 plt.ylabel('Missclassification')
56626 plt.xlabel('Number of neighbours')
56727 plt.show()
56828 """
56929
57030
57131
57232 # creating the model
57333 model = skl_nb.KNeighborsClassifier(n_neighbors = 120, weights = '
57434     distance')
57535
57636
57737 # Scaling the data, otherwise
57838 scaler = pp.StandardScaler().fit(X)
57939 model.fit(scaler.transform(X),Y)
58040 y_hat = model.predict(scaler.transform(X_test))
58141
58242
58343
58444 '''
58545 # oskalad data
58646 model.fit(X,Y)
58747 y_hat = model.predict(X_test)'''
58848
58949 # Get confusion matrix
59050 diff = pd.crosstab(y_hat, Y_test)
59151 print(f'Confusion matrix: \n {diff}')
59252
59353 # No. of TP,TN,FP,FN
59454 '''TP = diff.iloc[0,0]
59555 TN = diff.iloc[1,1]
59656 FP = diff.iloc[1,0]
59757 FN = diff.iloc[0,1]'''
59858
59959 # Get metrics:
60060 print(skl_m.classification_report(Y_test, y_hat))

```

Listing 4: Code for K- nearest neighbours

```

601 1 import numpy as np
602 2 import pandas as pd
603 3 import matplotlib.pyplot as plt
604 4 import sklearn.linear_model as skl_lm
605 5 import sklearn.preprocessing as pp
606 6 import sklearn.metrics as skl_m
607 7
608 8 df = pd.read_csv('training_data_vt2025.csv')
609 9 #df.info()
610 10
611 11 # Modify the dataset, emphasizing different variables
612 12 #df.iloc[:,12]=df.iloc[:,12]**2
613 13 #df.iloc[:,13]=np.sqrt(df.iloc[:,13])
614 14 #df.iloc[:,11] = df.iloc[:,11]**2
615 15
616 16 df['month_cos'] = np.cos(df.month*np.pi/12)
617 17 df['month_sin'] = np.sin(df.month*np.pi/12)
618 18
619 19 # time of day, replaed with low,medium and high demand,
620 20 # adding the new categories back in the end.
621 21 def categorize_demand(hour):
622 22     if 20 <= hour or 7 >= hour:
623 23         return 'night'
624 24     elif 8 <= hour <= 14:
625 25         return 'day'
626 26     elif 15 <= hour <= 19:
627 27         return 'evening'
628 28
629 29 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
630 30 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
631 31 drop_first=False)
632 31 df = pd.concat([df, df_dummies], axis=1)
633 32
634 33 # converting to bools
635 34 def if_zero(data):
636 35     if data == 0:
637 36         return True
638 37     else:
639 38         return False
640 39
641 40 # temperature
642 41
643 42 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
644 43 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
645 44
646 45 # Split into train and test:
647 46
648 47 #df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
649 48 #df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
650 49 np.random.seed(0)
651 50
652 51 df_modified=df[['holiday',
653 52                'weekday',
654 53                'summertime',
655 54                'temp',
656 55                'dew',
657 56                'humidity',
658 57                'visibility',
659 58                'windspeed',
660 59                'month_cos',
661 60                'month_sin',
662 61                'demand_day',
663 62                'demand_evening',
664 63                'demand_night',
665 64                'snowdepth_bool',

```

```

66655         'precip_bool',
66766         'increase_stock'']]
66867
66968 N = df_modified.shape[0]
67069 n = round(0.7*N)
67170 trainI = np.random.choice(N,size=n,replace=False)
67271 trainIndex = df_modified.index.isin(trainI)
67372 train = df_modified.iloc[trainIndex]
67473 test = df_modified.iloc[~trainIndex]
67574
67675 # Set up X,Y
67776
67877 # Train data
67978 X = train.iloc[:,0:-2]
68079 Y = train['increase_stock']
68180
68281 # Test data
68382 X_test = test.iloc[:,0:-2]
68483 Y_test = test['increase_stock']
68584
68685 model = skl_lm.LogisticRegression()
68786
68887 # Scaling the data, otherwise
68988 scaler = pp.StandardScaler().fit(X)
69089 model.fit(scaler.transform(X),Y)
69190 y_hat = model.predict(scaler.transform(X_test))
69291
69392 '''
69493 # oskalad data
69594 model.fit(X,Y)
69695 y_hat = model.predict(X_test)'''
69796
69897 # Get confusion matrix
69998 diff = pd.crosstab(y_hat, Y_test)
70099 print(f'Confusion matrix: \n {diff}')
70100
70201 # No. of TP,TN,FP,FN
70302 '''TP = diff.iloc[0,0]
70403 TN = diff.iloc[1,1]
70504 FP = diff.iloc[1,0]
70605 FN = diff.iloc[0,1]'''
70706
70807 # Get metrics:
70908 print(skl_m.classification_report(Y_test, y_hat))

```

Listing 5: Code for Logistic Regression