

---

# 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

Here we review the theoretical background of the models. We follow mostly [1].

Let  $\{\mathbf{x}_i, y_i\}_{i \in \mathbb{N}}$  be the training data set, where  $\mathbf{x}_i$  is a matrix representing the input, and  $y_i$  is the output.

### 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

$$X\theta = y$$

given by the normal equations

$$X^T X \theta = X^T y$$

where  $X$  is the training data matrix,  $\theta$  is the coefficient vector and  $y$  is the training output. The parameter vector is then used in the sigmoid function  $\sigma(z) : \mathbb{R} \rightarrow [0, 1]$

$$\sigma(z) = \frac{e^z}{1 + e^z} :$$

where  $z = x^T \theta$ , and  $x$  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. The model was tuned using the `lbfgs` algorithm, which is standard in *SKLearn*. It optimizes the weights  $\theta_i$  to minimize the log-loss function:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y_i \log(h_{\theta}(x_i)) + (1 - y_i) \log(1 - h_{\theta}(x_i))] + \frac{\lambda}{2} \|\theta\|^2, \quad (1)$$

where  $\lambda$  is the regularization strength (default for *SKLearn*:  $\lambda = 1.0$ ),  $h_{\theta}$  is the sigmoid function for each of the  $m$  input vectors  $x_i$ , and  $\theta$  is the parameter vector being optimized. It is being optimized through a gradient method.

#### 2.1.2 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.

### 51 2.1.3 Non-parametric method: k-Nearest Neighbour

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

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

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

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

63 where  $y_i$  is the class of the nearest neighbour,  $\chi$  is the characteristic function

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

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

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

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

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

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

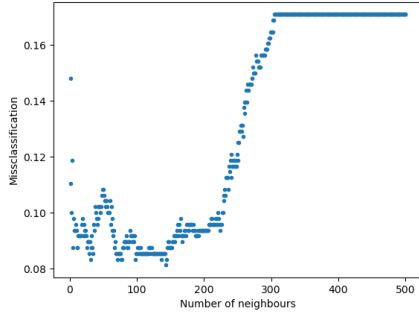
### 84 2.1.4 Discriminant analysis: LDA and QDA

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

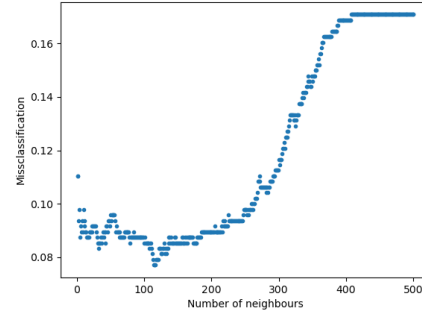
$$p(y|\mathbf{x}) = \frac{p(y, \mathbf{x})}{p(\mathbf{x})} = \frac{p(y)p(\mathbf{x}|y)}{\int_y p(y, \mathbf{x})}$$

89 For the discrete version it is obtained:

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



(a) Uniform distance test for  $k$ .



(b) Weighted distance test for  $k$ .

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

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:

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

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

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

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

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

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

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

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

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

## 2.2 Input Data Modification

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

- *Kept as-is:* weekday, windspeed, visibility, temp
- *Modified:*

- month - split into two inputs, one cosine and one sine part. This make the new inputs linear and can follow the fluctuations of the year. The original input was discarded.
- hour\_of\_day - split into three boolean variables: demand\_day, demand\_evening, and demand\_night, reflecting if the time was between 08-14, 15-19 or 20-07 respectively. This was done because plotting the data showed three different plateaus of demand for the different time intervals. The original input was discarded.
- snowdepth, precip were transformed into booleans, reflecting if it was raining or if there was snow on the ground or not. This was done as there was no times where demand was high when it was raining or when there was snow on the ground.
- *Removed:* cloudcover, day\_of\_week, snow, dew, holiday, summertime. These were removed due to being redundant (e.g. summertime), not showing a clear trend (e.g. 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

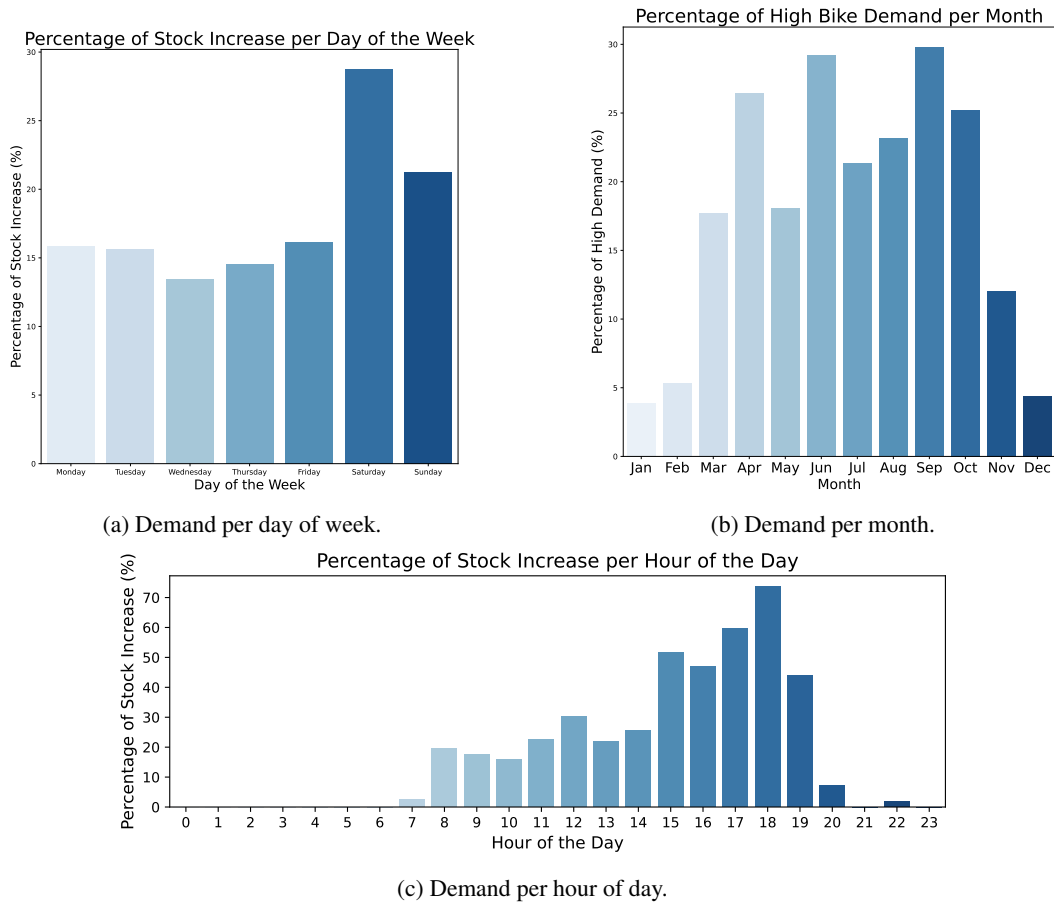


Figure 2: Bike demand vs. day of week and month.

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,

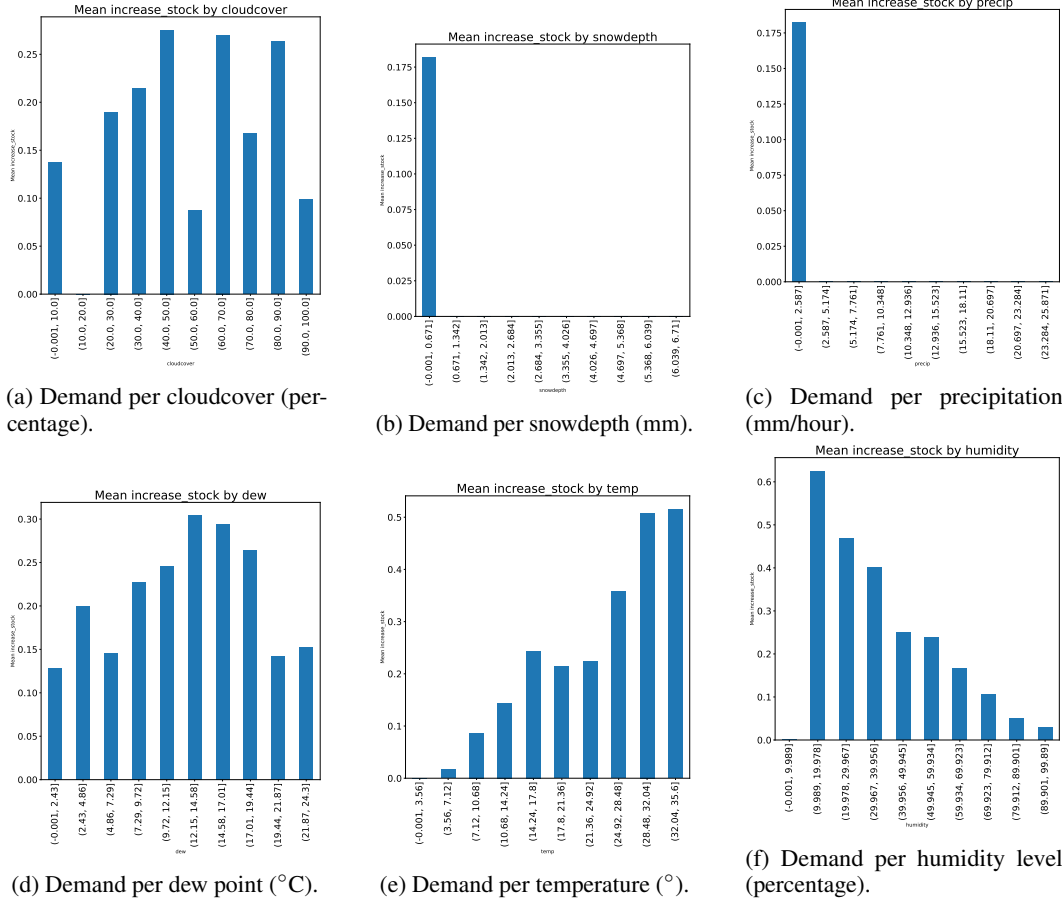


Figure 3: Bike demand vs. various weather parameters.

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.

## 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%

153  
154 outperforming linear and quadratic regression on accuracy, precision and recall. Out of random forest  
155 and kNN the group would proceed with the kNN method, its higher accuracy and precision score out  
156 waying the slightly better recall score of random forest. This will mean a slight loss in income caused  
157 by increasing false negatives but is thought to be covered by fewer false positives.

## 158 5 Conclusion

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

162  
163 One reason for the discriminant analysis falling short of the other models is likely due to  
164 these models being designed with the assumption of variables being normally distributed. This is not  
165 the case for this particular data set.

## 166 References

167 [1] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön. *Machine learning: a first course for*  
168 *engineers and scientists*. Cambridge University Press, 2022.

## 169 A Appendix

```

170 1 import pandas as pd
171 2 import numpy as np
172 3 from sklearn.model_selection import train_test_split
173 4 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
174 5 from sklearn.linear_model import LogisticRegression
175 6 from sklearn.metrics import accuracy_score
176 7 from sklearn.metrics import classification_report
177 8
178 9 df = pd.read_csv('training_data_vt2025.csv')
179 0
180 1 # modify the month to represent the periodicity that is observed in
181   data.
182 2 df['month_cos'] = np.cos(df['month']*2*np.pi/12)
183 3 df['month_sin'] = np.sin(df['month']*2*np.pi/12)
184 4
185 5 # time of day, replaced with 3 bool values: is_night, is_day and
186   is_evening,
187 6 # adding the new categories back in the end.
188 7 def categorize_demand(hour):
189 8     if 20 <= hour or 7 >= hour:
190 9         return 'night'
191 0     elif 8 <= hour <= 14:
192 1         return 'day'
193 2     elif 15 <= hour <= 19:
194 3         return 'evening'
195 4
196 5 df['time_of_day'] = df['hour_of_day'].apply(categorize_demand)
197 6 df_dummies = pd.get_dummies(df['time_of_day'], prefix='is', drop_first
198   =False)
199 7 df = pd.concat([df, df_dummies], axis=1)
200 8
201 9 # Create bool of snowdepth and percipitation
202 0 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
203 1 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
204 2
205 3 # Seperate training data from target
206 4 X=df[['#holiday',
207 5     'weekday',
208 6     '#summertime',
209 7     'temp',
210 8     '#dew',
211 9     '#humidity',
212 0     '#visibility',
213 1     '#windspeed',
214 2     '#month',
215 3     'month_cos',
216 4     'month_sin',
217 5     '#hour_of_day',
218 6     'is_day',
219 7     'is_evening',
220 8     'is_night',
221 9     '#hour_cos',
222 0     '#hour_sin',
223 1     'snowdepth_bool',
224 2     'precip_bool'
225 3     ]]
226 4
227 5 y=df['increase_stock']
228 6
229 7 # Split dataset into training and test sets
230 8 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
231   =0.2, random_state=42)
232 9

```



```

23360 # Apply Linear Discriminant Analysis (LDA)
23461 lda = LinearDiscriminantAnalysis(n_components=1)
23562 X_train_lda = lda.fit_transform(X_train, y_train)
23663 X_test_lda = lda.transform(X_test)
23764
23865 # Train a classifier (Logistic Regression)
23966 clf = LogisticRegression()
24067 clf.fit(X_train_lda, y_train)
24168
24269 # Make predictions
24370 y_pred = clf.predict(X_test_lda)
24471
24572 # Evaluate accuracy
24673 accuracy = accuracy_score(y_test, y_pred)
24774 print(f"Model Accuracy: {accuracy:.2f}")
24875
24976 print(classification_report(y_test, y_pred))

```

Listing 1: Code for LDA

```

250 1 import pandas as pd
251 2 import numpy as np
252 3 from sklearn.model_selection import train_test_split
253 4 from sklearn.discriminant_analysis import
254     QuadraticDiscriminantAnalysis
255 5 from sklearn.metrics import accuracy_score
256 6 from sklearn.metrics import classification_report
257 7
258 8 df = pd.read_csv('training_data_vt2025.csv')
259 9
26010 # modify the month to represent the periodicity that is observed in
261    data.
26211 df['month_cos'] = np.cos(df['month']*2*np.pi/12)
26312 df['month_sin'] = np.sin(df['month']*2*np.pi/12)
26413
26514 # time of day, replaced with 3 bool values: is_night, is_day and
266    is_evening,
26715 # adding the new categories back in the end.
26816 def categorize_demand(hour):
26917     if 20 <= hour or 7 >= hour:
27018         return 'night'
27119     elif 8 <= hour <= 14:
27220         return 'day'
27321     elif 15 <= hour <= 19:
27422         return 'evening'
27523
27624 df['time_of_day'] = df['hour_of_day'].apply(categorize_demand)
27725 df_dummies = pd.get_dummies(df['time_of_day'], prefix='is', drop_first
278    =False)
27926 df = pd.concat([df, df_dummies], axis=1)
28027
28128 # Create bool of snowdepth and percipitation
28229 df['snowdepth_bool'] = df['snowdepth'].where(df['snowdepth'] == 0, 1)
28330 df['precip_bool'] = df['precip'].where(df['precip'] == 0, 1)
28431
28532 # Seperate training data from target
28633 X=df[['#holiday',
28734     'weekday',
28835     '#summertime',
28936     'temp',
29037     '#dew',
29138     '#humidity',
29239     '#visibility',
29340     '#windspeed',
29441     '#month',

```

```

29542         'month_cos',
29643         'month_sin',
29744         #'hour_of_day',
29845         'is_day',
29946         'is_evening',
30047         'is_night',
30148         #'snowdepth_bool',
30249         'precip_bool'
30350     ]]
30451
30552 y=df['increase_stock']
30653
30754 # Split dataset into training and test sets
30855 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
309         =0.2, random_state=42)
31056
31157 # Apply Quadratic Discriminant Analysis (QDA)
31258 qda = QuadraticDiscriminantAnalysis()
31359 X_train_lda = qda.fit(X_train, y_train)
31460
31561 # Make predictions
31662 y_pred = qda.predict(X_test)
31763
31864 # Evaluate accuracy
31965 accuracy = accuracy_score(y_test, y_pred)
32066 print(f"Model Accuracy: {accuracy:.2f}")
32167
32268 print(classification_report(y_test, y_pred))

```

Listing 2: Code for QDA

```

323 1 import pandas as pd
324 2 import numpy as np
325 3 import matplotlib
326 4 import matplotlib.pyplot as plt
327 5 from sklearn import tree
328 6 from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
329 7 import graphviz
330 8 from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
331 9 from sklearn.metrics import classification_report
33210
33311 df = pd.read_csv('training_data_vt2025.csv')
33412 #df.info()
33513
33614 # Modify the dataset, emphasizing different variables
33715 df.iloc[:,12]=df.iloc[:,12]**2
33816 df.iloc[:,13]=np.sqrt(df.iloc[:,13])
33917 df.iloc[:,11] = df.iloc[:,11]**2
34018
34119 df['month_cos'] = np.cos(df.month*np.pi/12)
34220 df['month_sin'] = np.sin(df.month*np.pi/12)
34321
34422 # time of day, replaed with low,medium and high demand,
34523 # adding the new categories back in the end.
34624 def categorize_demand(hour):
34725     if 20 <= hour or 7 >= hour:
34826         return 'night'
34927     elif 8 <= hour <= 14:
35028         return 'day'
35129     elif 15 <= hour <= 19:
35230         return 'evening'
35331
35432 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
35533 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
356         drop_first=False)

```

```

35734 df = pd.concat([df, df_dummies], axis=1)
35835
35936 # converting to bools
36037 def if_zero(data):
36138     if data == 0:
36239         return True
36340     else:
36441         return False
36542
36643 # temperature
36744
36845 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
36946 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
37047
37148 # Split into train and test:
37249
37350 df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
37451 df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
37552 np.random.seed(0)
37653
37754 df_modified=df[ ['#holiday',
37855                  'weekday',
37956                  '#summertime',
38057                  'temp',
38158                  '#dew',
38259                  '#humidity',
38360                  'visibility',
38461                  'windspeed',
38562                  'month_cos',
38663                  'month_sin',
38764                  'demand_day',
38865                  'demand_evening',
38966                  'demand_night',
39067                  'snowdepth_bool',
39168                  'precip_bool',
39269                  'increase_stock']]
39370
39471 N = df_modified.shape[0]
39572 n = round(0.7*N)
39673 trainI = np.random.choice(N,size=n,replace=False)
39774 trainIndex = df_modified.index.isin(trainI)
39875 train = df_modified.iloc[trainIndex]
39976 test = df_modified.iloc[~trainIndex]
40077
40178 X_train = train.drop(columns=['increase_stock'])
40279 # Need to transform the qualitative variables to dummy variables
40380
40481 y_train = train['increase_stock']
40582
40683 model = RandomForestClassifier(random_state=42)
40784 param_grid = {
40885     'n_estimators': [100, 200, 300],
40986     'max_depth': [10, 20, None],
41087     'min_samples_split': [2, 5, 10],
41188     'min_samples_leaf': [1, 2, 4]
41289 }
41390
41491 # Set up Grid Search
41592 random_search = RandomizedSearchCV(model, param_grid, cv=5, scoring='
41693     accuracy', n_jobs=-1, verbose=2)
41794
41895 # Fit on training data
41996 random_search.fit(X_train, y_train)
42097
42198 # Get the best hyperparameters

```

```

42298 print("Best Parameters: ", random_search.best_params_)
42399 print("Best Accuracy: %.2f" % random_search.best_score_)
42400
42501 # Update the model with the best parameters
42602 best_model = random_search.best_estimator_
42703
42804 # Fit the best model on the training data
42905 best_model.fit(X_train, y_train)
43006
43107 # Make predictions using the optimized model
43208
43309
43410
43511
43612 ###
43713 #dot_data = tree.export_graphviz(model, out_file=None, feature_names =
438      X_train.columns, class_names = model.classes_,
43914 #      filled=True, rounded=True,
440      leaves_parallel=True, proportion=True)
44115 #graph = graphviz.Source(dot_data)
44216 #graph.render("decision_tree", format="pdf")
44317 X_test = test.drop(columns=['increase_stock'])
44418 y_test = test['increase_stock']
44519 y_predict = best_model.predict(X_test)
44620
44721
44822
44923 print(classification_report(y_test, y_predict))

```

Listing 3: Code for Random Forest

```

4501 import numpy as np
4512 import pandas as pd
4523 import matplotlib.pyplot as plt
4534 import sklearn.linear_model as skl_lm
4545 import sklearn.preprocessing as pp
4556 import sklearn.metrics as skl_m
4567
4578 import sklearn.neighbors as skl_nb
4589
4590 df = pd.read_csv('training_data_vt2025.csv')
4601 #df.info()
4612
4623 # Modify the dataset, emphasizing different variables
4634 #df.iloc[:,12]=df.iloc[:,12]**2
4645 #df.iloc[:,13]=np.sqrt(df.iloc[:,13])
4656 #df.iloc[:,11] = df.iloc[:,11]**2
4667
4678 df['month_cos'] = np.cos(df.month*np.pi/12)
4689 df['month_sin'] = np.sin(df.month*np.pi/12)
4690
4701 # time of day, replaed with low,medium and high demand,
4712 # adding the new categories back in the end.
4723 def categorize_demand(hour):
4734     if 20 <= hour or 7 >= hour:
4745         return 'night'
4756     elif 8 <= hour <= 14:
4767         return 'day'
4778     elif 15 <= hour <= 19:
4789         return 'evening'
4790
4801 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
4812 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
482     drop_first=False)
4833 df = pd.concat([df, df_dummies], axis=1)

```

```

48434
48535 # converting to bools
48636 def if_zero(data):
48737     if data == 0:
48838         return True
48939     else:
49040         return False
49141
49242 # temperature
49343
49444 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
49545 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
49646
49747 # Split into train and test:
49848
49949 #df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
50050 #df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
50151 np.random.seed(0)
50252
50353 df_modified=df[['holiday',
50454                 'weekday',
50555                 '#summertime',
50656                 'temp',
50757                 '#dew',
50858                 'humidity',
50959                 'visibility',
51060                 'windspeed',
51161                 'month_cos',
51262                 'month_sin',
51363                 'demand_day',
51464                 'demand_evening',
51565                 'demand_night',
51666                 'snowdepth_bool',
51767                 'precip_bool',
51868                 'increase_stock']]
51969
52070 N = df_modified.shape[0]
52171 n = round(0.7*N)
52272 trainI = np.random.choice(N,size=n,replace=False)
52373 trainIndex = df_modified.index.isin(trainI)
52474 train = df_modified.iloc[trainIndex]
52575 test = df_modified.iloc[~trainIndex]
52676
52777 # Set up X,Y
52878
52979 # Train data
53080 X = train.iloc[:,0:-2]
53181 Y = train['increase_stock']
53282
53383 # Test data
53484 X_test = test.iloc[:,0:-2]
53585 Y_test = test['increase_stock']
53686
53787
53888 """
53989 # Tests for k-value
54090 # TEST 1 - uniform distance
54191 missclassification = []
54292 for k in range(500): # Try n_neighbours = 1, 2, ...,
54393
54494     #kNN method
54595     scaler = pp.StandardScaler().fit(X)
54696     model = skl_nb.KNeighborsClassifier(n_neighbors = k+1, weights = '
547     uniform')
54897     model.fit(scaler.transform(X),Y)

```

```

5498
5509     # Prediction
5510     y_hat = model.predict(scaler.transform(X_test))
5521     missclassification.append(np.mean(y_hat != Y_test))
5532
5543 K = np.linspace(1, 500, 500)
5554 plt.plot(K, missclassification, '.')
5565 plt.ylabel('Missclassification')
5576 plt.xlabel('Number of neighbours')
5587 plt.show()
5598
5609 #TEST 2
5610 missclassification = []
5621 for k in range(500): # Try n_neighbours = 1, 2, ...,
5632
5643     #kNN method
5654     scaler = pp.StandardScaler().fit(X)
5665     model = skl_nb.KNeighborsClassifier(n_neighbors = k+1, weights = '
567     distance')
5686     model.fit(scaler.transform(X),Y)
5697
5708     # Prediction
5719     y_hat = model.predict(scaler.transform(X_test))
5720     missclassification.append(np.mean(y_hat != Y_test))
5731
5742 K = np.linspace(1, 500, 500)
5753 plt.plot(K, missclassification, '.')
5764 plt.ylabel('Missclassification')
5775 plt.xlabel('Number of neighbours')
5786 plt.show()
5797 """
5808
5819
5820
5831 # creating the model
5842 model = skl_nb.KNeighborsClassifier(n_neighbors = 120, weights = '
585     distance')
5863
5874
5885 # Scaling the data, otherwise
5896 scaler = pp.StandardScaler().fit(X)
5907 model.fit(scaler.transform(X),Y)
5918 y_hat = model.predict(scaler.transform(X_test))
5929
5930
5941
5952 '''
5963 # oskalad data
5974 model.fit(X,Y)
5985 y_hat = model.predict(X_test)'''
5996
6007 # Get confusion matrix
6018 diff = pd.crosstab(y_hat, Y_test)
6029 print(f'Confusion matrix: \n {diff}')
6030
6041 # No. of TP,TN,FP,FN
6052 '''TP = diff.iloc[0,0]
6063 TN = diff.iloc[1,1]
6074 FP = diff.iloc[1,0]
6085 FN = diff.iloc[0,1]'''
6096
6107 # Get metrics:
6118 print(skl_m.classification_report(Y_test, y_hat))

```

Listing 4: Code for K- nearest neighbours

```

612 1 import numpy as np
613 2 import pandas as pd
614 3 import matplotlib.pyplot as plt
615 4 import sklearn.linear_model as skl_lm
616 5 import sklearn.preprocessing as pp
617 6 import sklearn.metrics as skl_m
618 7
619 8 df = pd.read_csv('training_data_vt2025.csv')
620 9 #df.info()
621 10
622 11 # Modify the dataset, emphasizing different variables
623 12 #df.iloc[:,12]=df.iloc[:,12]**2
624 13 #df.iloc[:,13]=np.sqrt(df.iloc[:,13])
625 14 #df.iloc[:,11] = df.iloc[:,11]**2
626 15
627 16 df['month_cos'] = np.cos(df.month*2*np.pi/12) # period of 12 months
628 17 df['month_sin'] = np.sin(df.month*2*np.pi/12)
629 18
630 19 # time of day, replaed with low,medium and high demand,
631 20 # adding the new categories back in the end.
632 21 def categorize_demand(hour):
633 22     if 20 <= hour or 7 >= hour:
634 23         return 'night'
635 24     elif 8 <= hour <= 14:
636 25         return 'day'
637 26     elif 15 <= hour <= 19:
638 27         return 'evening'
639 28
640 29 # Adding the categories back, but creating three new categories
641 30 # for the different times
642 31 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
643 32 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
644 33 drop_first=False)
645 34 df = pd.concat([df, df_dummies], axis=1)
646 35
647 36 # converting to bools
648 37 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
649 38 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
650 39
651 40 # Split into train and test:
652 41 np.random.seed(0)
653 42
654 43 # Can try different combinations, which inputs give worse perf etc
655 44 df_modified=df[['holiday',
656 45                 'weekday',
657 46                 'summertime',
658 47                 'temp',
659 48                 'dew',
660 49                 'humidity',
661 50                 'visibility',
662 51                 'windspeed',
663 52                 'month_cos',
664 53                 'month_sin',
665 54                 'demand_day',
666 55                 'demand_evening',
667 56                 'demand_night',
668 57                 'snowdepth_bool',
669 58                 'precip_bool',
670 59                 'increase_stock']]
671 60
672 61 N = df_modified.shape[0]
673 62 n = round(0.7*N)
674 63 trainI = np.random.choice(N,size=n,replace=False)
675 64 trainIndex = df_modified.index.isin(trainI)
676 65 train = df_modified.iloc[trainIndex]

```

```

67755 test = df_modified.iloc[~trainIndex]
67856
67957 # Set up X,Y
68058
68159 # Train data
68260 X = train.iloc[:,0:-2]
68371 Y = train['increase_stock']
68472
68573 # Test data
68674 X_test = test.iloc[:,0:-2]
68775 Y_test = test['increase_stock']
68876
68977 model = skl_lm.LogisticRegression()
69078
69179 # Scaling the data, otherwise
69280 scaler = pp.StandardScaler().fit(X)
69381 model.fit(scaler.transform(X),Y)
69482 y_hat = model.predict(scaler.transform(X_test))
69583
69684 '''
69785 # oskalad data
69886 model.fit(X,Y)
69987 y_hat = model.predict(X_test)'''
70088
70189 # Get confusion matrix
70290 diff = pd.crosstab(y_hat, Y_test)
70391 print(f'Confusion matrix: \n {diff}')
70492
70593 # No. of TP,TN,FP,FN
70694 '''TP = diff.iloc[0,0]
70795 TN = diff.iloc[1,1]
70896 FP = diff.iloc[1,0]
70997 FN = diff.iloc[0,1]'''
71098
71199 # Get metrics:
71200 print(skl_m.classification_report(Y_test, y_hat))

```

Listing 5: Code for Logistic Regression