# CA5 - Herman Ellingsen

April 28, 2022

## 1 DAT200 CA5 2022

Kaggle username: Herman Ellingsen

#### 1.0.1 Imports

[252]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from os import path

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier

from sklearn.impute import SimpleImputer
from sklearn.impute import KNNImputer

from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.base import BaseEstimator, TransformerMixin
```

#### 1.0.2 Reading data

[253]:
```
raw_train_data = pd.read_pickle("/Users/Herman/Documents/dat200-ca5-2022/train.
 ↪pkl")
raw_test_data = pd.read_pickle("/Users/Herman/Documents/dat200-ca5-2022/test.
 ↪pkl")
```

#### 1.0.3 Data exploration and visualisation

[254]:
```
raw_train_data.info()  # Meta data about the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12165 entries, 0 to 12164
```

```
Data columns (total 13 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   Season                             12165 non-null  object
 1   Year                               12165 non-null  object
 2   Month                              12165 non-null  object
 3   Hour                               12165 non-null  object
 4   Holiday                            12165 non-null  object
 5   Weekday                            12165 non-null  object
 6   Working day                        12165 non-null  object
 7   Weather situation                  12165 non-null  object
 8   Temperature (normalized)           12165 non-null  object
 9   Feels-like temperature (normalized)  12165 non-null  object
 10  Humidity (normalized)              12165 non-null  object
 11  Windspeed                          12165 non-null  object
 12  Rental bikes count                 12165 non-null  int64
dtypes: int64(1), object(12)
memory usage: 1.2+ MB
```

[265]:
```
raw_train_data.describe() # Meta data about the dataset
# Dont know why it only shows one column?
```

[265]:
```
       Rental bikes count
count        12165.000000
mean           189.081381
std            181.511771
min              1.000000
25%             39.000000
50%            141.000000
75%            280.000000
max            976.000000
```

[266]:
```
raw_train_data.shape # Checking the shape of the data
```

[266]:
```
(12165, 13)
```

[267]:
```
raw_train_data.head() # Print the first 5 rows to have a look at the data
```

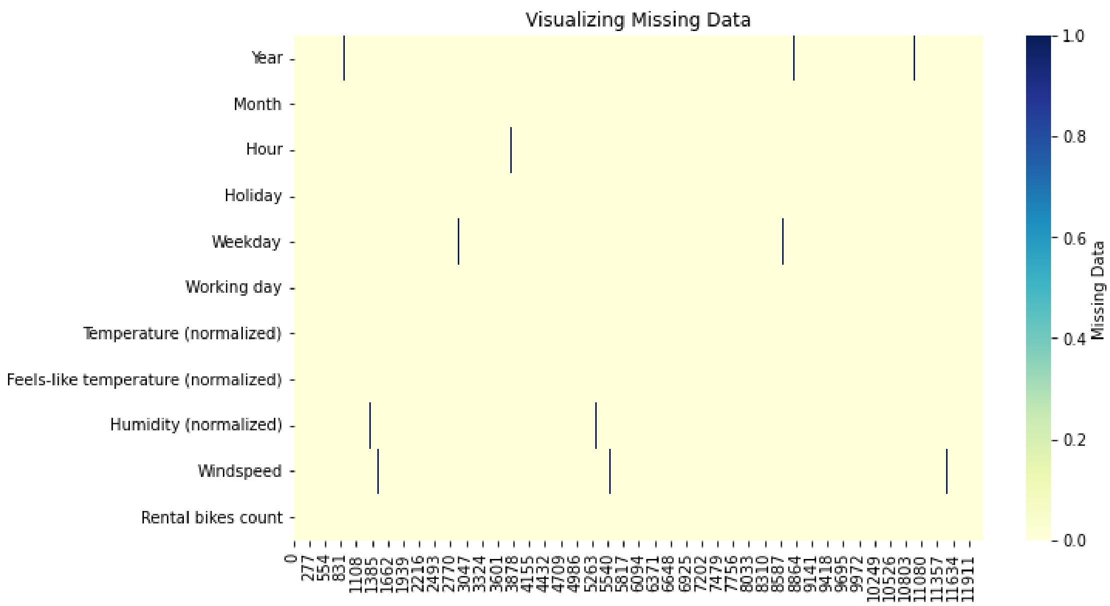
[267]:
```
   Season Year Month Hour Holiday Weekday Working day       Weather situation  \
0  Summer    1     6   18       0       1           1  Clear or partly cloudy
1    Fall    1    10   11       0       3           1     Misty and/or cloudy
2  Spring    0     6   22       0       6           0  Clear or partly cloudy
3  Spring    0     3   21       0       2           1     Misty and/or cloudy
4    Fall    1    11    5       0       2           1     Misty and/or cloudy

  Temperature (normalized) Feels-like temperature (normalized)  \
0                     0.76                              0.6667
1                     0.36                              0.3485
```

```
2                      0.64                                0.6212
3                      0.42                                0.4242
4                      0.34                                0.3333

   Humidity (normalized) Windspeed  Rental bikes count
0                   0.27    0.4478                 791
1                   0.66    0.2239                 189
2                   0.57    0.2239                 190
3                   0.54    0.2836                  87
4                   0.66    0.1343                  34
```

[268]:
```
# Explore the missing values

print(raw_train_data.isin(['missing']).sum())

numerical_data = raw_train_data.drop(columns={"Season", "Weather situation"})
numerical_data = numerical_data.replace("missing", np.nan)


plt.figure(figsize=(10,6))
sns.heatmap(numerical_data.isna().transpose(),
            cmap="YlGnBu",
            cbar_kws={'label': 'Missing Data'})
plt.title("Visualizing Missing Data")
```

```
Season                                   39
Year                                     37
Month                                    31
Hour                                     50
Holiday                                  30
Weekday                                  36
Working day                              34
Weather situation                        27
Temperature (normalized)                 36
Feels-like temperature (normalized)      24
Humidity (normalized)                    30
Windspeed                                34
Rental bikes count                        0
dtype: int64
```

[268]:
```
Text(0.5, 1.0, 'Visualizing Missing Data')
```




### 1.0.4 Data cleaning

[269]:
```python
# Cleaning dataset using KNN-imputer:
numerical_data = raw_train_data.drop(columns={"Season", "Weather situation"})
numerical_data = numerical_data.replace("missing", np.nan)

imp_mean = KNNImputer(n_neighbors=3)
imp_mean.fit(numerical_data)
data = pd.DataFrame(imp_mean.transform(numerical_data), columns=numerical_data.columns)

samlet_data = pd.concat([data, raw_train_data[["Season", "Weather situation"]]], axis=1)

y_knnimputer = samlet_data["Rental bikes count"]
X_knnimputer = samlet_data.drop(columns={"Rental bikes count"})

X_knnimputer.isna().values.any()
```

[269]:
```
False
```

[270]:
```python
# Cleaning dataset using SimpleImputer:
numerical_data = raw_train_data.drop(columns={"Season", "Weather situation"})
numerical_data = numerical_data.replace("missing", np.nan)

imp_mean = SimpleImputer(strategy="mean")
imp_mean.fit(numerical_data)
```

```
data = pd.DataFrame(imp_mean.transform(numerical_data), columns=numerical_data.
 ↪columns)

samlet_data = pd.concat([data, raw_train_data[["Season", "Weather␣
 ↪situation"]]], axis=1)


y_simpleimputer = samlet_data["Rental bikes count"]
X_simpleimputer = samlet_data.drop(columns={"Rental bikes count"})

X_simpleimputer.isna().values.any()
```

[270]: False

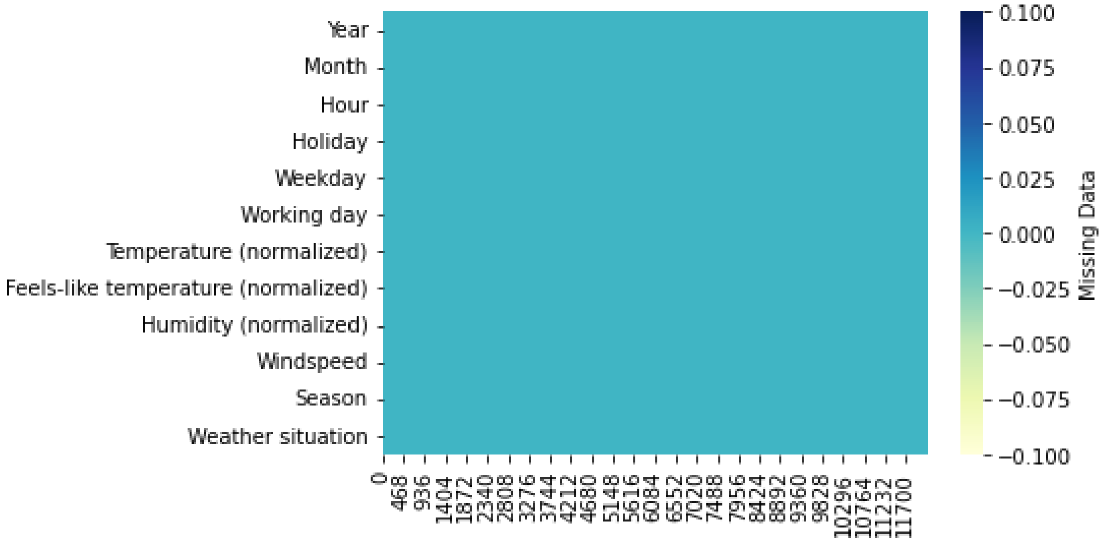
[271]:
```
# Cleaning dataset by simply removing null values:
X2 = raw_train_data.replace("missing", np.nan)
X2 = X2.dropna()

y_dropna = X2["Rental bikes count"]
X_dropna = X2.drop(columns={"Rental bikes count"})

X_dropna.isna().values.any()
```

[271]: False

### 1.0.5 Data exploration after cleaning

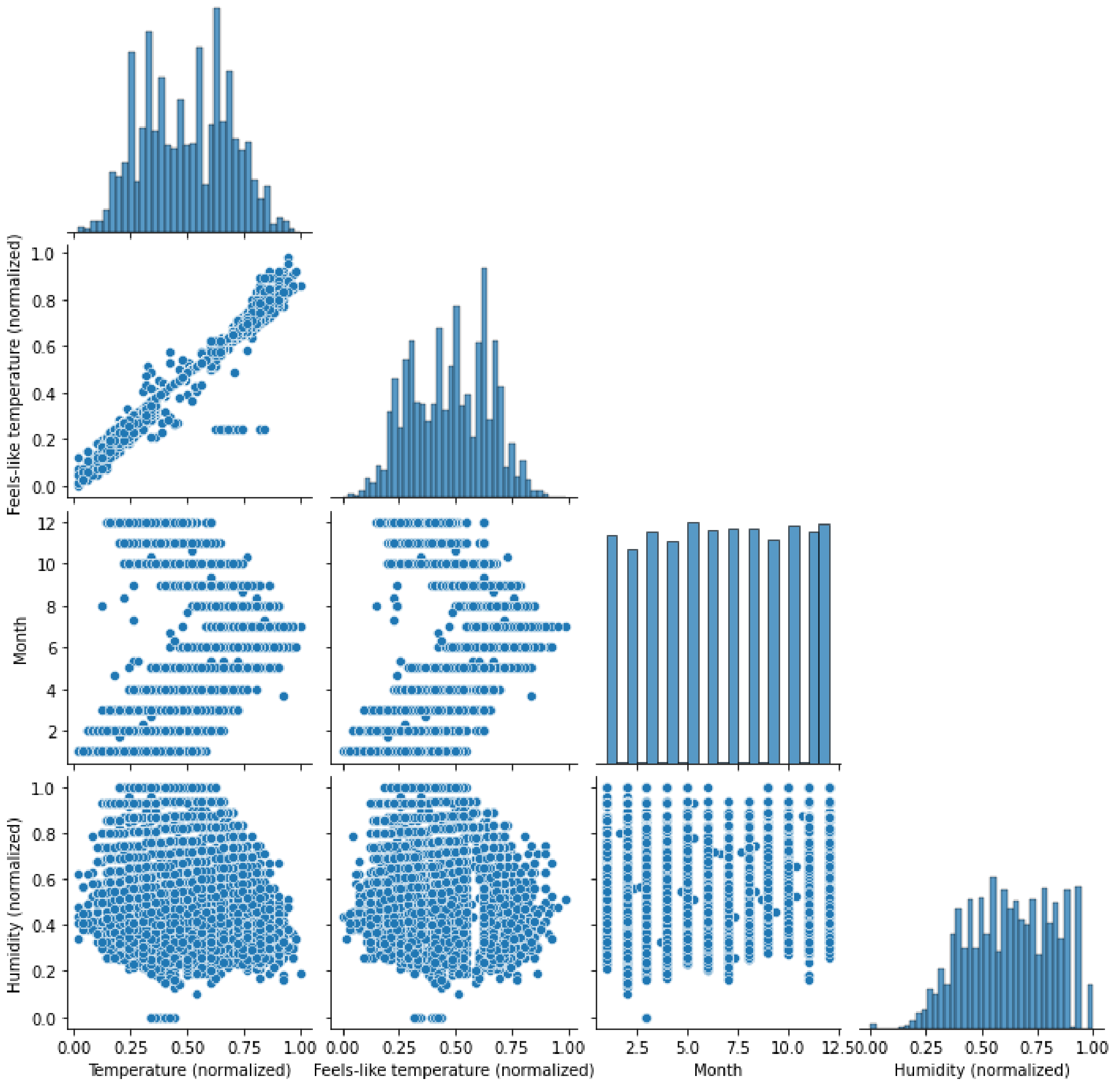
[272]:
```
# Checking for any missing values after cleaning
sns.heatmap(X_knnimputer.isna().transpose(),
            cmap="YlGnBu",
            cbar_kws={'label': 'Missing Data'})

print(X_knnimputer.isna().any())
```

```
Year                                    False
Month                                   False
Hour                                    False
Holiday                                 False
Weekday                                 False
Working day                             False
Temperature (normalized)                False
Feels-like temperature (normalized)     False
Humidity (normalized)                   False
Windspeed                               False
Season                                  False
Weather situation                       False
dtype: bool
```



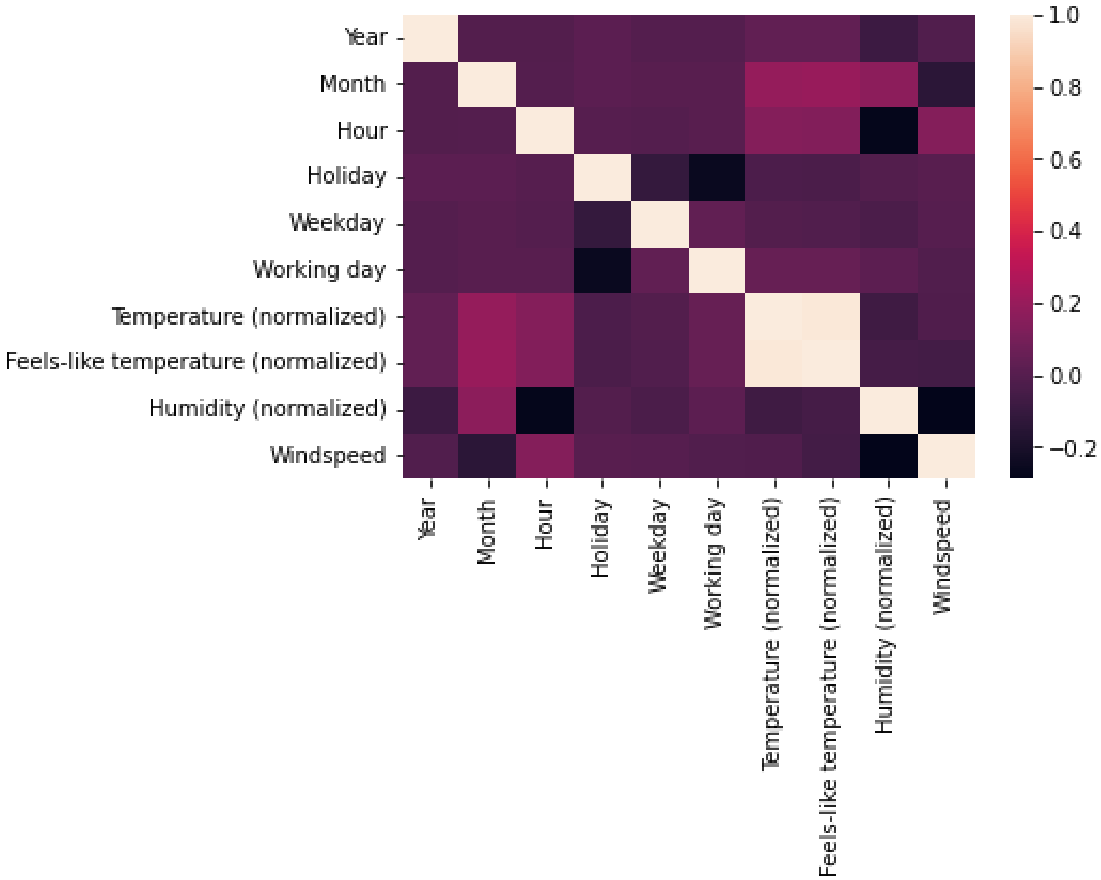

[273]:
```
# Compare features to each other
# Kept the features that was most interesting to explore
sns.pairplot(data=X_knnimputer[["Temperature (normalized)", "Feels-like␣
↪temperature (normalized)", "Month", "Humidity (normalized)"]], corner=True)
```

[273]: <seaborn.axisgrid.PairGrid at 0x7faa31087190>



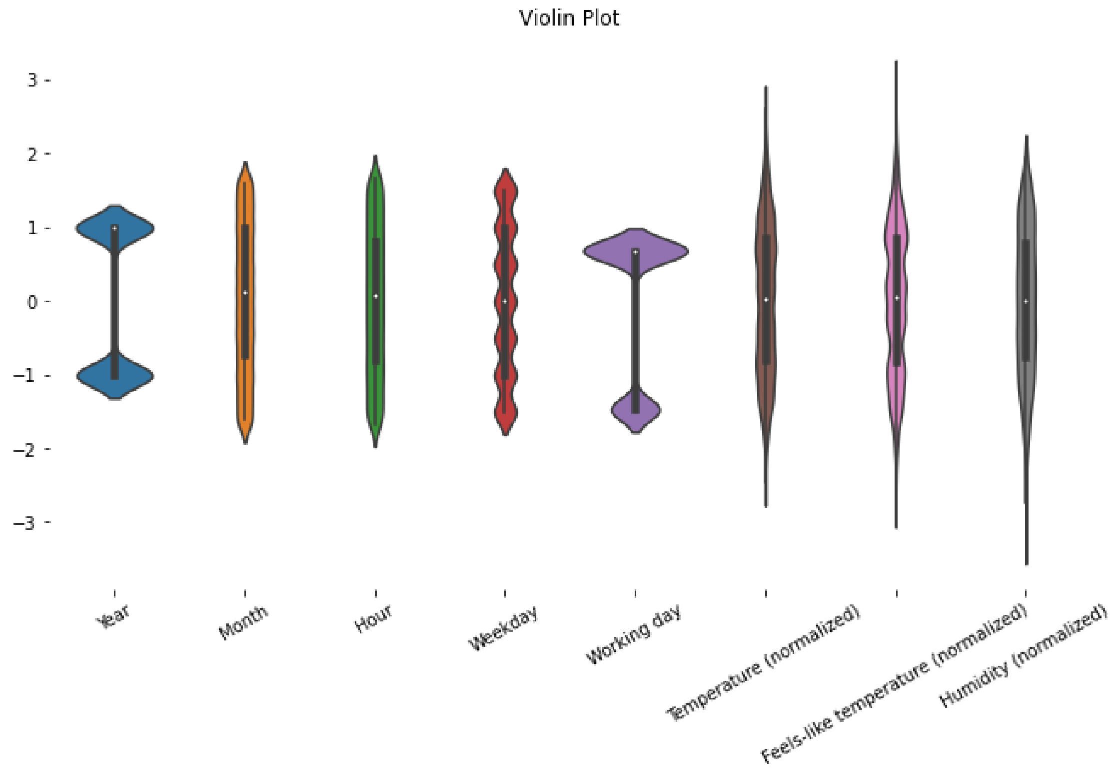

[274]:
```
# Use a heatmap to see the features correlation
# with each other
sns.heatmap(X_knnimputer.corr())
```

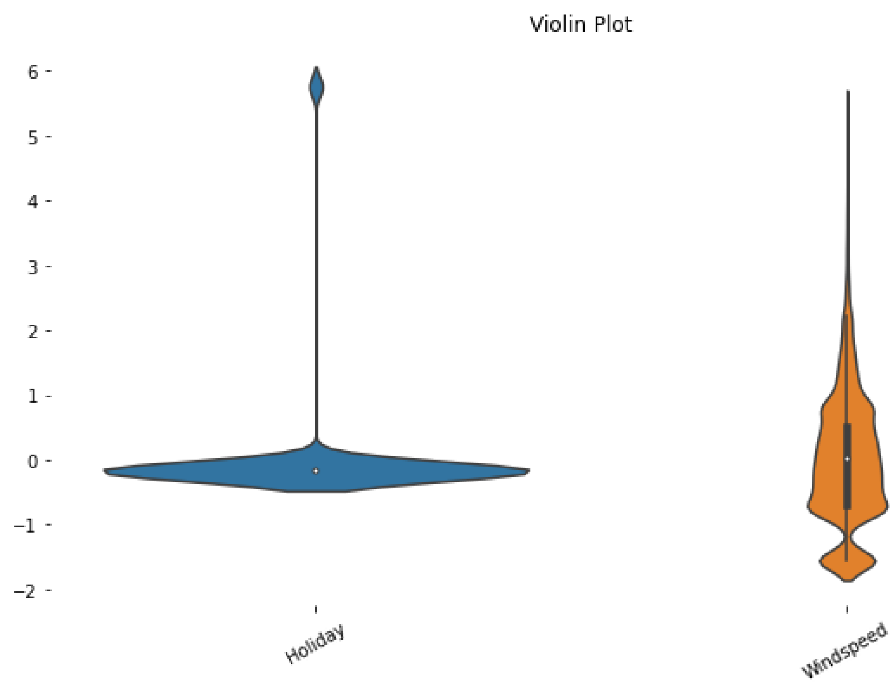
[274]: <AxesSubplot:>




[275]:
```
# Use violinplot to show distribution of values in each feature
# After scaling the data, its easier to see distribution
data_4_violin_1 = X_knnimputer.drop(columns={"Season", "Weather situation",␣
↪"Holiday", "Windspeed"})
data_4_violin_2 = X_knnimputer[["Holiday", "Windspeed"]]
scaler = StandardScaler()
data_4_violin_sc_1 = scaler.fit_transform(data_4_violin_1.
↪select_dtypes(include='number'))
data_4_violin_sc_2 = scaler.fit_transform(data_4_violin_2.
↪select_dtypes(include='number'))

for subset, col in zip([data_4_violin_sc_1, data_4_violin_sc_2],
                       [data_4_violin_1, data_4_violin_2]):
    f, ax = plt.subplots(figsize=(11, 6))
    sns.violinplot(data=subset)
    sns.despine(left=True, bottom=True)
    plt.xticks(ticks=range(0, len(col.columns)), labels=col.columns,␣
↪rotation=30)
```

```
plt.title("Violin Plot")
plt.show()
```



### 1.0.6 Data preprocessing

[283]:
```
# Preprocessing class to be used in pipeline
class Preprocessing(BaseEstimator, TransformerMixin):
    # initializer
    def __init__(self):
        pass

    def fit(self, X, y = None):
        return self

    def transform(self, X, y = None):

        X_testern = X.copy().dropna()

        X_testern = X_testern.drop(columns={"Windspeed"})
        X_testern["Year_sqrt"] = np.sqrt(X_testern["Year"]+1)
        X_testern["Year_log"] = np.log(X_testern["Year"]+1)
        X_testern["Temperature (normalized)_log"] = np.
↪log(X_testern["Temperature (normalized)"]+1)
        X_testern["Humidity (normalized)_log"] = np.log(X_testern["Humidity␣
↪(normalized)"]+1)
        X_testern["Year"] = (X_testern["Year"]+1)**2 # Remove?
        X_testern["Year_3"] = (X_testern["Year"]+1)**3
        X_testern["Year_4"] = (X_testern["Year"]+1)**4
        X_testern["Hour * Working day"] = (X_testern["Hour"]) *␣
↪X_testern["Working day"]
        X_testern["Hour * Month"] = X_testern["Hour"] * X_testern["Month"]
        X_testern["hum/hol"] = ((X_testern["Humidity (normalized)"]+1) /␣
↪(X_testern["Holiday"]+1)**2)


        X_testern["Working day * Holiday"] = (X_testern["Working day"]+1) /␣
↪(X_testern["Holiday"]+1)
        X_testern["Month * Holiday"] = (X_testern["Month"]+1) *␣
↪(X_testern["Holiday"]+1)
        X_testern["Month * Humidity"] = (X_testern["Month"]+1) /␣
↪(X_testern["Humidity (normalized)"]+1)

        return X_testern
```

### 1.0.7 Modelling

**Data pipeline with regression model**

```
[284]: for X, y, imp in zip([X_knnimputer, X_simpleimputer, X_dropna],
                     [ y_knnimputer, y_simpleimputer, y_dropna],
                     ["KNNImputer()",  "SimpleImputer", "dropna()"]):


    # Here I concatinate the train and test set before I make dummy variables.
    # This is because I got different amount of dummy variables if I
    # did this seperate
    train_objs_num = len(X)
    dataset = pd.concat(objs=[X, raw_test_data], axis=0)
    dataset_preprocessed = pd.get_dummies(dataset)
    train_preprocessed = dataset_preprocessed[:train_objs_num]
    test_preprocessed = dataset_preprocessed[train_objs_num:]


    pipe_rf = make_pipeline(Preprocessing(),
                            StandardScaler(),
                            RandomForestRegressor(random_state=2,
                                                  n_jobs=-1))




    # Set the value range for the GridSearch
    # I tried different value ranges, but did not have time
    # to run through the whole thing again. So this only
    # a short version with the hyperparamters I
    # used in the final submission
    param_grid_rf = [{'randomforestregressor__n_estimators': [350],
                      'randomforestregressor__max_depth': [25],
                      'randomforestregressor__min_samples_leaf': [1],
                      'randomforestregressor__min_samples_split': [2],
                      'randomforestregressor__min_impurity_decrease': [0.0035],
                      'randomforestregressor__max_features': [0.57]}]




    # Create the GridSearch
    gs_rf = GridSearchCV(estimator=pipe_rf,
                         param_grid=param_grid_rf,
                         scoring='r2',
                         cv=5,
                         refit=True,
                         n_jobs=-1)



    # Fit and predict
    gs_rf.fit(train_preprocessed, y)
```

```python
# Print the best scores and the hyperparameters that gave the best results
print("-"*60)
print(f"R2 score and best parameters when using {imp}")
print("-"*60)
print(f"R2 Score: {gs_rf.best_score_}\n")
print("{:<50} {:<15}\n".format("Hyperparameter", "Value"))
for key, value in gs_rf.best_params_.items():
    print(f"{key:<50} {value:<15}")
print(" ")
print(" ")
#estimator.get_params().keys()
#clf.named_steps["preprocessing"].transform(X_train)
```

```
------------------------------------------------------------
R2 score and best parameters when using KNNImputer()
------------------------------------------------------------
R2 Score: 0.9443797224466651

Hyperparameter                                     Value

randomforestregressor__max_depth                   25
randomforestregressor__max_features                0.57
randomforestregressor__min_impurity_decrease       0.0035
randomforestregressor__min_samples_leaf            1
randomforestregressor__min_samples_split           2
randomforestregressor__n_estimators                350


------------------------------------------------------------
R2 score and best parameters when using SimpleImputer
------------------------------------------------------------
R2 Score: 0.9426547926541382

Hyperparameter                                     Value

randomforestregressor__max_depth                   25
randomforestregressor__max_features                0.57
randomforestregressor__min_impurity_decrease       0.0035
randomforestregressor__min_samples_leaf            1
randomforestregressor__min_samples_split           2
randomforestregressor__n_estimators                350


------------------------------------------------------------
R2 score and best parameters when using dropna()
------------------------------------------------------------
R2 Score: 0.9475818374524299
```

```
Hyperparameter                                      Value

randomforestregressor__max_depth                    25
randomforestregressor__max_features                 0.57
randomforestregressor__min_impurity_decrease        0.0035
randomforestregressor__min_samples_leaf             1
randomforestregressor__min_samples_split            2
randomforestregressor__n_estimators                 350
```

**Data pipeline with classification model** Binning train target values

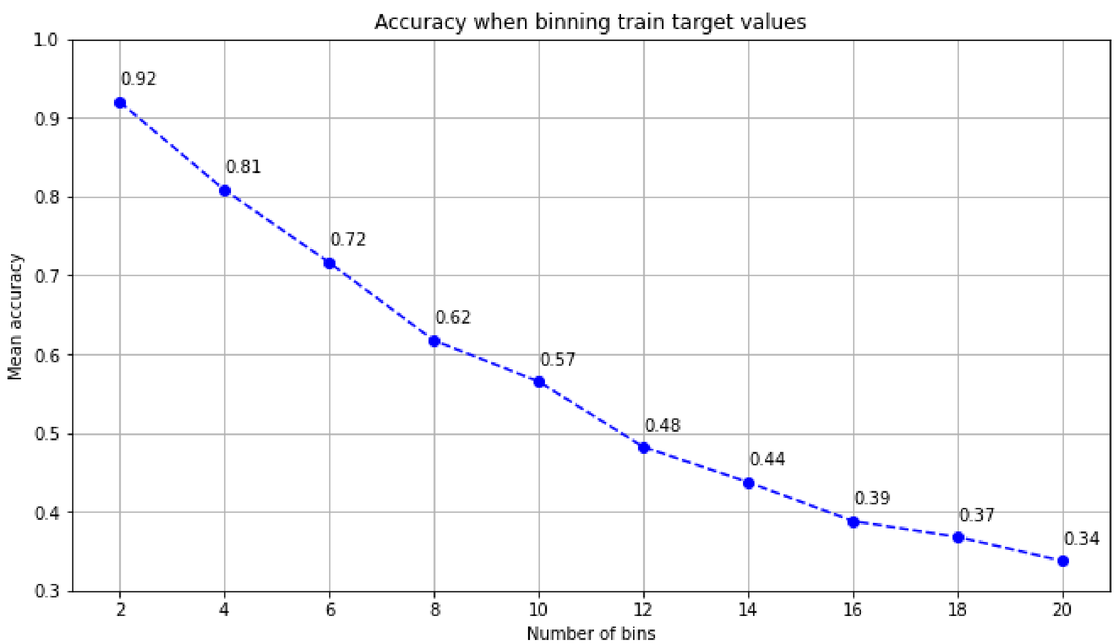
Can be performed with ex. pandas.qcut or pandas.cut

```
n_bins = 10
y_train_binned = pd.cut(y_train, n_bins, labels=False) # or
y_train_binned = pd.qcut(y_train, n_bins, labels=False)
```

[279]:
```
n_bins = range(2, 21, 2)
results = []


for bins in n_bins:

    y = samlet_data["Rental bikes count"]
    y_binned = pd.qcut(y, bins, labels=False)
    X = samlet_data.drop(columns={"Rental bikes count"})
    X_testern = pd.get_dummies(X.copy())

    X_train, X_test, y_train, y_test = train_test_split(X_testern, y_binned,␣
 ↪test_size=0.20, random_state=2)


    # Make pipeline for model
    pipe_classification = make_pipeline(Preprocessing(),
                                         RandomForestClassifier(random_state=2))

    pipe_classification.fit(X_train, y_train)

    results.append(pipe_classification.score(X_test, y_test))
```

[280]:
```
f, ax = plt.subplots(figsize=(11, 6))
plt.plot(n_bins, results, "bo--")
plt.title("Accuracy when binning train target values")
plt.xlabel("Number of bins")
plt.xticks(ticks=n_bins, labels=n_bins)
```

```
plt.ylabel("Mean accuracy")
plt.grid()

axes = plt.gca()
axes.set_ylim([0.30, 1])

for x, y in zip(n_bins, results):
    label = "{:.2f}".format(y)

    plt.annotate(label,
                 (x, y),
                 textcoords="offset points",
                 xytext=(0, 10),
                 ha='left')
```




As we see from the plot above, the mean accuracy decreases with the number of bins. I believe this is because the classifier needs to be more accurate when the number of target classes increases, therefore making it harder to get a high accuracy.

Note: For simplicity the test above was done on one train_test_split. The the "true" result might therefore be splightly different.

### 1.0.8 Kaggle submission

[281]:
```
def submission_file(y_pred):
    pathern = "/Users/Herman/Documents/CA5_submissions/CA5_01.csv"
    if not path.exists(pathern):
        pass
    else:
        while path.exists(pathern):
            pathern = "/Users/Herman/Documents/CA5_submissions/
↪CA5_"+str(int(pathern[-6:-4])+1).zfill(2)+".csv"
            print(pathern)

    CA5_sub = pd.DataFrame() # Make empty dataframe for submission
    y_pred_df = pd.DataFrame(y_pred)
    CA5_sub["idx"] = y_pred_df.index # Insert index into the submission df
    CA5_sub["Rental bikes count"] = y_pred # Insert the predictions into the␣
↪submission df
    CA5_sub.to_csv(pathern, index=None) # Convert dataframe into csv-file
    test_csv = pd.read_csv(pathern) # Checking if its in the right format
    print(test_csv)
```

[282]:
```
y_pred = gs_rf.predict(pd.get_dummies(test_preprocessed))
submission_file(y_pred)
```

```
/Users/Herman/Documents/CA5_submissions/CA5_02.csv
/Users/Herman/Documents/CA5_submissions/CA5_03.csv
/Users/Herman/Documents/CA5_submissions/CA5_04.csv
/Users/Herman/Documents/CA5_submissions/CA5_05.csv
/Users/Herman/Documents/CA5_submissions/CA5_06.csv
/Users/Herman/Documents/CA5_submissions/CA5_07.csv
/Users/Herman/Documents/CA5_submissions/CA5_08.csv
/Users/Herman/Documents/CA5_submissions/CA5_09.csv
/Users/Herman/Documents/CA5_submissions/CA5_10.csv
/Users/Herman/Documents/CA5_submissions/CA5_11.csv
/Users/Herman/Documents/CA5_submissions/CA5_12.csv
/Users/Herman/Documents/CA5_submissions/CA5_13.csv
/Users/Herman/Documents/CA5_submissions/CA5_14.csv
/Users/Herman/Documents/CA5_submissions/CA5_15.csv
/Users/Herman/Documents/CA5_submissions/CA5_16.csv
/Users/Herman/Documents/CA5_submissions/CA5_17.csv
/Users/Herman/Documents/CA5_submissions/CA5_18.csv
/Users/Herman/Documents/CA5_submissions/CA5_19.csv
/Users/Herman/Documents/CA5_submissions/CA5_20.csv
/Users/Herman/Documents/CA5_submissions/CA5_21.csv
/Users/Herman/Documents/CA5_submissions/CA5_22.csv
/Users/Herman/Documents/CA5_submissions/CA5_23.csv
/Users/Herman/Documents/CA5_submissions/CA5_24.csv
       idx  Rental bikes count
```

```
0        0          550.686667
1        1            6.324912
2        2          143.388000
3        3          469.266667
4        4            4.311955
…     …                  …
5209  5209          291.237619
5210  5210           38.746667
5211  5211           93.905000
5212  5212           42.424000
5213  5213          710.860000

[5214 rows x 2 columns]
```