B365 project: Bike Sharing Demand Prediction

0. Dataset introduction

This project is based on a fairly new dataset called Seoul Bike Sharing Demand Data Set (2020-3-11). Here is the source url: https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand# (https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand) Since this dataset is published March this year, it is within expectation that little reference can be found on the internet. So, we though it is a good chance to practice what we have learned in class on this dataset on our own.

These are the attributes(14) for this dataset:
Date : year-month-day
Rented Bike count - Count of bikes rented at each hour
Hour - Hour of he day
Temperature-Temperature in Celsius
Humidity - %
Windspeed - m/s
Visibility - 10m
Dew point temperature - Celsius
Solar radiation - MJ/m2
Rainfall - mm
Snowfall - cm
Seasons - Winter, Spring, Summer, Autumn
Holiday - Holiday/No holiday
Functional Day - NoFunc(Non Functional Hours), Fun(Functional hours)

We need to predict the number of bikes rented per hour. This is clearly a regression problem, so the models we choose are linear regression, polynomial regression, ridge regression, DecisionTreeRegressor, RandomForestRegressor and KNeighborsRegressor. We plan to use 5 cross validation and grid search on these models and compare them based on r2 score. Additionally, we will self implement linear regression with 5 cross validation use Normal equation method. We will then compare that result with the linear regression model from sklearn library to prove our hypothesis that normal equation and gradient descent yields the same result is correct.

1. Import Packages

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
```

2. Import Seoul Bike Sharing Demand Data Set

In [2]:

```
df = pd.read_csv("SeoulBikeData.csv",encoding= 'unicode_escape')
df.head()
```

Out[2]:

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6

In [3]:
```
df.dtypes
```

Out[3]:
```
Date                          object
Rented Bike Count              int64
Hour                           int64
Temperature(°C)              float64
Humidity(%)                    int64
Wind speed (m/s)             float64
Visibility (10m)               int64
Dew point temperature(°C)    float64
Solar Radiation (MJ/m2)      float64
Rainfall(mm)                 float64
Snowfall (cm)                float64
Seasons                       object
Holiday                       object
Functioning Day               object
dtype: object
```

In [4]:
```
df.isnull().sum() # no null data
```

Out[4]:
```
Date                         0
Rented Bike Count            0
Hour                         0
Temperature(°C)              0
Humidity(%)                  0
Wind speed (m/s)             0
Visibility (10m)             0
Dew point temperature(°C)    0
Solar Radiation (MJ/m2)      0
Rainfall(mm)                 0
Snowfall (cm)                0
Seasons                      0
Holiday                      0
Functioning Day              0
dtype: int64
```

3. Preprocessing and Analysis Dataset

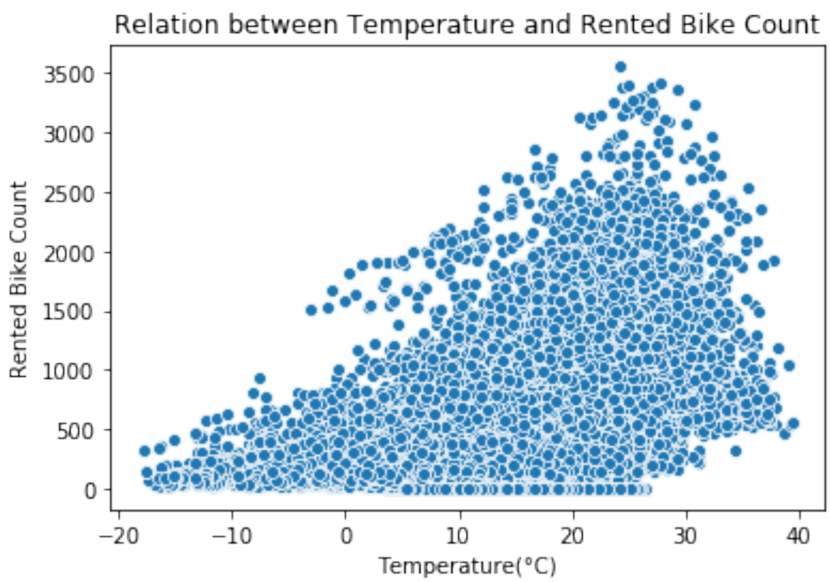
Since "Seasons", "Holiday" and "Functioning Day" are categorical variables, we need to change them to one hot econding.

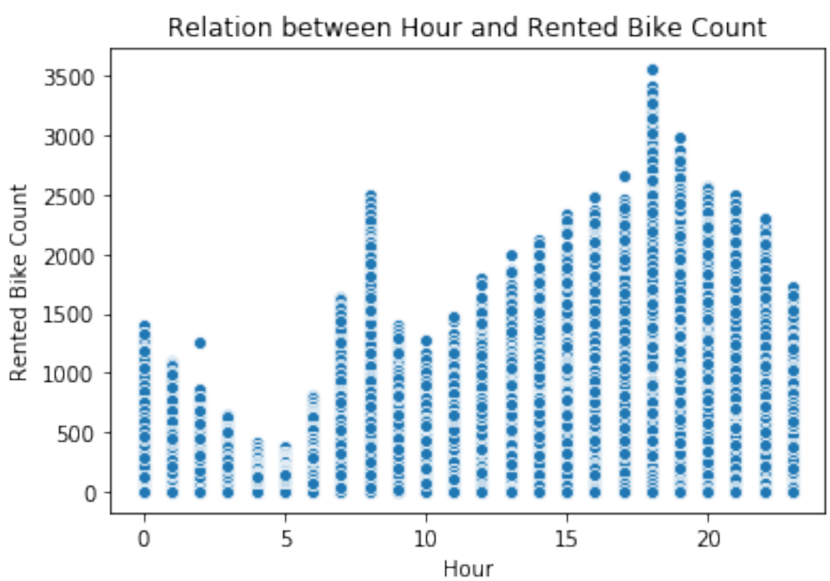
In [5]:
```
df = df.drop(["Date"],axis=1)
df = pd.get_dummies(df, drop_first=True)
```

In [6]:
```
print(df.corr()['Rented Bike Count'].sort_values())
```

```
Seasons_Winter              -0.424925
Humidity(%)                 -0.199780
Snowfall (cm)               -0.141804
Rainfall(mm)                -0.123074
Seasons_Spring               0.022888
Holiday_No Holiday           0.072338
Wind speed (m/s)             0.121108
Visibility (10m)             0.199280
Functioning Day_Yes          0.203943
Solar Radiation (MJ/m2)      0.261837
Seasons_Summer               0.296549
Dew point temperature(°C)    0.379788
Hour                         0.410257
Temperature(°C)              0.538558
Rented Bike Count            1.000000
Name: Rented Bike Count, dtype: float64
```

In [7]:
```
plt.title('Relation between Temperature and Rented Bike Count')
sns.scatterplot(x=df['Temperature(°C)'],y=df['Rented Bike Count'])
plt.show()
plt.title('Relation between Hour and Rented Bike Count')
sns.scatterplot(x=df['Hour'],y=df['Rented Bike Count'])
plt.show()
```





In [8]:
```
# prepare X and y
X = df.drop(['Rented Bike Count'], axis = 1)
y = df['Rented Bike Count']
sc = StandardScaler()
X = sc.fit_transform(X)
```

4. Using grid search and 5 cross validation to find the best hypo parameters for Decision Tree and Random Forest.

In [9]:
```
DecTr_param = {
    "min_samples_split": [10, 15],
    "max_depth": [12, 14],
    "min_samples_leaf": [10],
    "max_leaf_nodes": [180, 210, 250],
}
DecTr_grid = GridSearchCV(DecisionTreeRegressor(), DecTr_param, cv=5)

DecTr_grid.fit(X,y)
best_DT = DecTr_grid.best_estimator_
best_DT # The best parameters
```

Out[9]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=14,
                      max_features=None, max_leaf_nodes=250,
                      min_impurity_decrease=0.0, min_impurity_split=N
one,
                      min_samples_leaf=10, min_samples_split=10,
                      min_weight_fraction_leaf=0.0, presort='deprecat
ed',
                      random_state=None, splitter='best')
```

In [10]:
```
RanForest_param = {
    'max_depth': [10,12],
    'min_samples_leaf': [2],
    'min_samples_split': [2],
    'n_estimators': [80,100]
}
RF_grid = GridSearchCV(RandomForestRegressor(), RanForest_param, cv=5)

RF_grid.fit(X,y.ravel())
best_RF = RF_grid.best_estimator_
best_RF
```

Out[10]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=12, max_features='auto', max_leaf_nod
es=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=2,
                      min_samples_split=2, min_weight_fraction_leaf=0
.0,
                      n_estimators=80, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

5. Self implement the linear regression with 5 cross validation.

Later we will find that normal equation yield the same result with gradient descent.

In [11]:
```
def mylinearRegression(X,y):
    # theta = (inv(X.T*X))*X.T*y
    dim = X.shape
    b=np.ones((dim[0],1))
    X = np.concatenate([X,b],axis=1)
    a = np.linalg.inv(np.dot(X.T,X))
    b = np.dot(X.T,y)
    theta = np.dot(a,b)
    return theta

def lrpredict(X,theta):
    dim = X.shape
    b=np.ones((dim[0],1))
    X = np.concatenate([X,b],axis=1)
    return np.dot(X,theta)

# metrics mean square error
def lrmse(yhat,y):
    num = len(y)
    error = np.sum((yhat-y)**2)/num
    return error

# metrics R_square:
# 1 - residual sum of square / total sum of squares
def lrr2(yhat,y):
    sse = np.sum((yhat - y)**2)
    sst = np.sum((y - y.mean())**2)
    r_square = 1 - (sse/sst)
    return r_square
```

In [12]:
```
from sklearn.model_selection import KFold
kf = KFold(n_splits = 5, random_state=1, shuffle=True)
my_lr_r2score = []
for train_idx, test_idx in kf.split(X, y):
    X_train, X_test, y_train, y_test = X[train_idx], X[test_idx], y[tr
    theta = mylinearRegression(X_train,y_train)
    yhat = lrpredict(X_test,theta)
    r2 = lrr2(yhat,y_test)
    my_lr_r2score.append(r2)
r2 = sum(my_lr_r2score)/len(my_lr_r2score)
print("Our self implemented linear regression with 5 fold cross valida
```

```
Our self implemented linear regression with 5 fold cross validation r
2 score is:
 0.548427216749686
```

6. Go through different regression models using sklearn library.

Though sklearn use gradient descent in linear regression to find theta, the idea behind it should be the same -- to minimize the MSE. The normal equation is more computational expansive since it inverse a matrix O(N^3). From the result, we saw our implementation of linear regression gives us the same answer with sklearn.

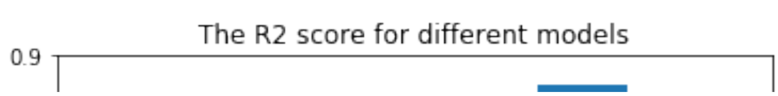
In [13]:
```
polyreg=make_pipeline(PolynomialFeatures(degree=2),LinearRegression())
lreg = LinearRegression()
ridge = Ridge(alpha=0.5)
KNN = KNeighborsRegressor()

lr = cross_val_score(lreg, X, y,scoring='r2', cv=kf)
lr = lr.mean()

polyr = cross_val_score(polyreg, X, y,scoring='r2', cv=kf)
polyr = polyr.mean()

rig = cross_val_score(ridge, X, y, scoring='r2', cv=kf)
rig = rig.mean()

dt = cross_val_score(best_DT, X, y,scoring='r2', cv=kf)
dt = dt.mean()

rf = cross_val_score(best_RF, X, y.ravel(),scoring='r2', cv=kf)
rf = rf.mean()

knn = cross_val_score(KNN, X, y,scoring='r2', cv=kf)
knn = knn.mean()
```

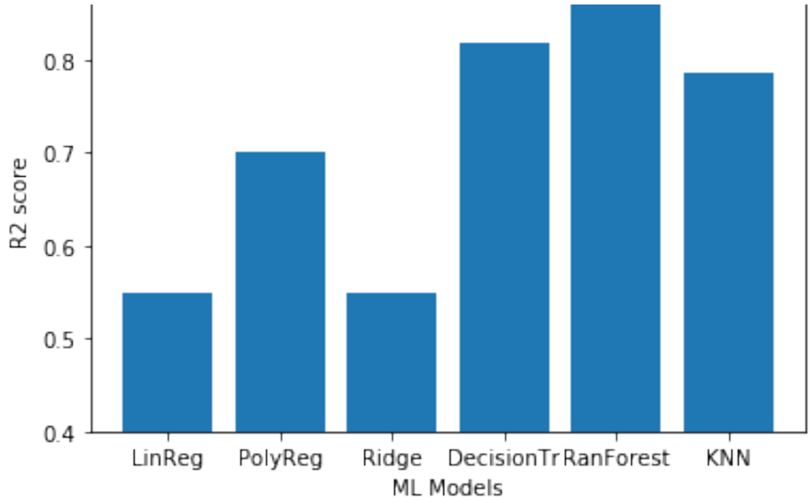
7. Compare different models using r2 metric.

From the graph we saw that different model perform largely different. Random Forest Regression gives the best result, and Linear Regression gives the worst.

In [14]:
```
methods = ['LinReg','PolyReg','Ridge','DecisionTr','RanForest', 'KNN']
scores=np.array([lr,polyr,rig,dt,rf,knn])
ind = [x for x,_ in enumerate(methods)]
plt.bar(ind, scores)
plt.xticks(ind, methods)
plt.xlabel("ML Models")
plt.ylabel("R2 score")
plt.ylim(0.4,0.9)
plt.title("The R2 score for different models")

plt.show
print("Our self implemented linear regression with 5 fold cross valida
print('\nThe r2 score for LinearRegression is:\n', lr)
print('\nThe r2 score for PloynomialRegression is:\n', polyr)
print('\nThe r2 score for DecisionTree is:\n', dt)
print('\nThe r2 score for RandomForest is:\n', rf)
print('\nThe r2 score for KNN is:\n', knn)
print('\nThe r2 score for Ridge Regression is:\n', rig)
```

```
Our self implemented linear regression with 5 fold cross validation r
2 score is:
 0.548427216749686

The r2 score for LinearRegression is:
 0.548427216749686

The r2 score for PloynomialRegression is:
 0.7005028838656342

The r2 score for DecisionTree is:
 0.8181992311356371

The r2 score for RandomForest is:
 0.8684334816398733

The r2 score for KNN is:
 0.785992040977317

The r2 score for Ridge Regression is:
 0.5484312051428192
```




0.8
0.7
0.6
0.5
0.4
R2 score
LinReg
PolyReg
Ridge
DecisionTr
RanForest
KNN
ML Models

In [ ]: