Ryan Long
DSC 680-T301
Project 1

In [1]:
```
#import libraries
import pandas as pd
import numpy as np
from pandas_profiling import ProfileReport
```

Data Import & Clean

In [2]:
```
data = pd.read_excel('data.xlsx')
```

In [3]:
```
# change time variable to timedelta, in minutes
data['TIME2'] = pd.to_timedelta(data['TIME'])/ np.timedelta64(1, 'm')
data['PACE_CALC2'] = pd.to_timedelta(data['PACE_CALC'])/ np.timedelta64(1, 'm')
```

In [4]:
```
#import
from sklearn import preprocessing

# label encode the date
le = preprocessing.LabelEncoder()
data['DATE_LABEL'] = le.fit_transform(data['DATE']) # 0-5 (beginning of season-er
data['COURSE_TYPE_LABEL'] = le.fit_transform(data['COURSE_TYPE']) #0=FLAT,1=HILLY
```

EDA

In [5]:
```
profile = ProfileReport(data,title="Pandas Profiling Report",explorative=True)
```

In [6]:
```
profile
```

Summarize dataset: 100% 54/54 [00:08<00:00, 7.17it/s, Completed]

Generate report structure: 100% 1/1 [00:04<00:00, 4.33s/it]

Render HTML: 100% 1/1 [00:01<00:00, 1.60s/it]

Overview

Dataset statistics

Number of variables	16
Number of observations	500
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	231.6 KiB
Average record size in memory	474.3 B

Variable types

Categorical	10
DateTime	1
Numeric	5

Alerts

`NAMETOKEN` has a high cardinality: 330 distinct values	**High cardinality**
`TIME` has a high cardinality: 336 distinct values	**High cardinality**

Out[6]:

In [7]:
```
#profile.to_widgets()
```

In [8]:
```
profile.to_file("Project_1_EDA.html")
```

Export report to file: 100% 1/1 [00:00<00:00, 30.30it/s]

In [9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   MEET_LOC           500 non-null    object
 1   DATE               500 non-null    datetime64[ns]
 2   PLACE              500 non-null    int64
 3   GRADE              500 non-null    int64
 4   NAMETOKEN          500 non-null    object
 5   TIME               500 non-null    object
 6   DIST_M             500 non-null    int64
 7   DIST_MI            500 non-null    float64
 8   PACE_CALC          500 non-null    object
 9   SCHOOL             500 non-null    object
 10  TEMP_F             500 non-null    int64
 11  COURSE_TYPE        500 non-null    object
 12  TIME2              500 non-null    float64
 13  PACE_CALC2         500 non-null    float64
 14  DATE_LABEL         500 non-null    int64
 15  COURSE_TYPE_LABEL  500 non-null    int32
dtypes: datetime64[ns](1), float64(3), int32(1), int64(5), object(6)
memory usage: 60.7+ KB
```

In [10]:
```
data
```

Out[10]:

	MEET_LOC	DATE	PLACE	GRADE	NAMETOKEN	TIME	DIST_M	DIST_I
0	Glenwood	2021-08-28	1	8	HebCa	00:13:53	2400	1
1	Glenwood	2021-08-28	2	8	HunMa	00:13:53	2400	1
2	Glenwood	2021-08-28	3	7	HugMe	00:14:10	2400	1
3	Glenwood	2021-08-28	4	7	BerGr	00:14:45	2400	1
4	Glenwood	2021-08-28	5	7	HesEl	00:14:49	2400	1
...	...	...	...	...	...	...	...	
495	NE_JH_State_XC_Meet_OPEN	2021-10-09	160	7	TutAd	00:20:23	3000	1
496	NE_JH_State_XC_Meet_OPEN	2021-10-09	161	8	McDTa	00:20:29	3000	1
497	NE_JH_State_XC_Meet_OPEN	2021-10-09	162	6	DalCa	00:20:31	3000	1
498	NE_JH_State_XC_Meet_OPEN	2021-10-09	163	7	KliRo	00:23:24	3000	1
499	NE_JH_State_XC_Meet_OPEN	2021-10-09	164	8	WicSa	00:23:53	3000	1

500 rows × 16 columns

Model Evaluation

Feature & Regression Review

In [11]:
```
# data just for reviewing
data = data[['DIST_M','PLACE','DATE_LABEL','TEMP_F','GRADE','TIME2','PACE_CALC2',
```

In [12]:
```
# set dependent and independent
X = data[['DIST_M','PLACE','DATE_LABEL','TEMP_F','GRADE','TIME2','COURSE_TYPE_LAE
y = data['PACE_CALC2']
```

In [13]:
```
#import library/module
from sklearn import linear_model
import statsmodels.api as sm
```

In [14]:
```
# with statsmodels
X = sm.add_constant(X) # adding a constant
```

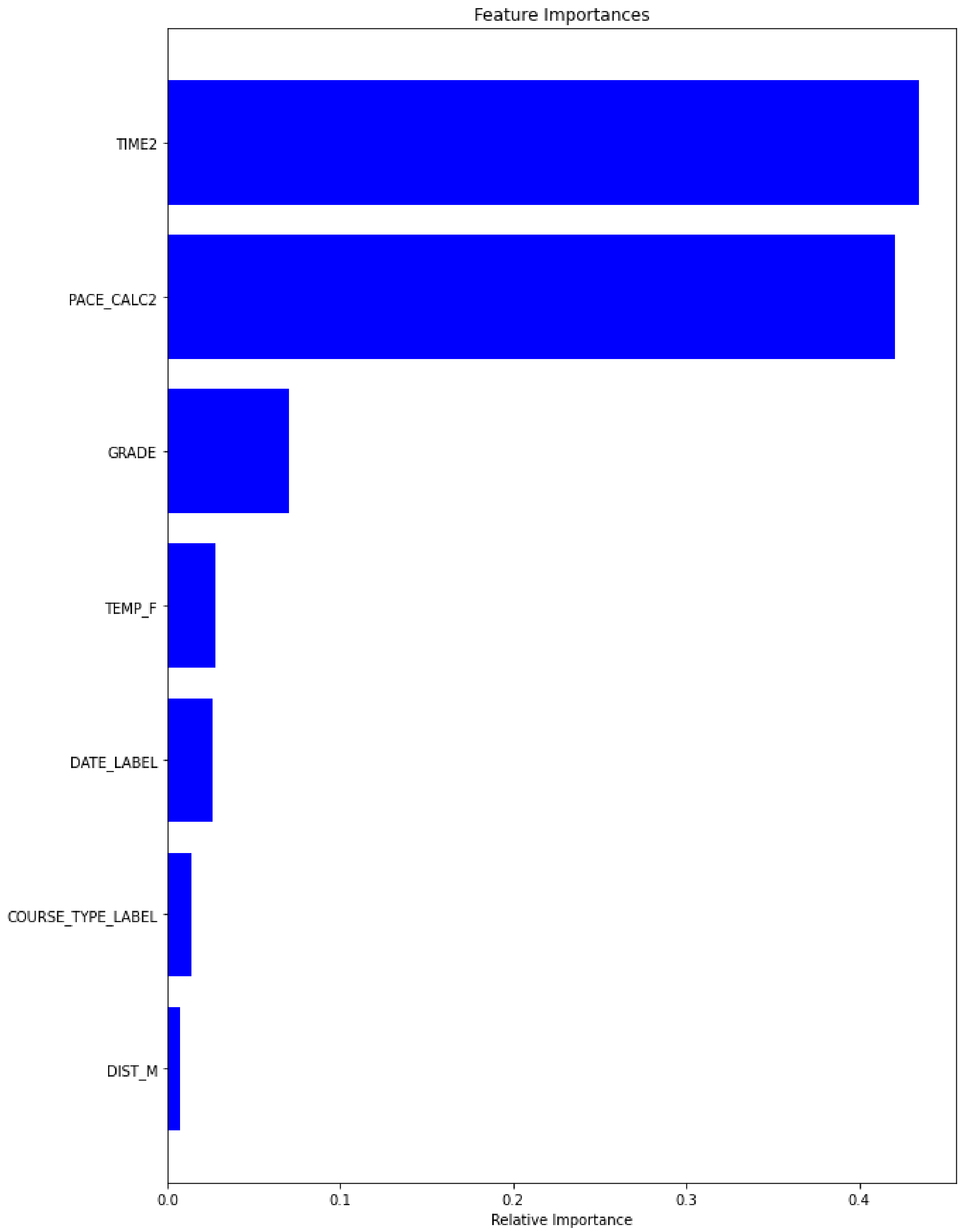
In [15]:
```
# fit & predict model
model = sm.OLS(y, X).fit()
predictions = model.predict(X)
```

In [16]:
```
#set model summary as variable to print
print_model = model.summary()
print(print_model)
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:             PACE_CALC2   R-squared:                       0.996
Model:                            OLS   Adj. R-squared:                  0.996
Method:                 Least Squares   F-statistic:                 1.869e+04
Date:                Fri, 01 Jul 2022   Prob (F-statistic):               0.00
Time:                        10:38:05   Log-Likelihood:                 344.78
No. Observations:                 500   AIC:                            -673.6
Df Residuals:                     492   BIC:                            -639.8
Df Model:                           7                                         
Covariance Type:            nonrobust                                         
===============================================================================
======
                       coef    std err          t      P>|t|      [0.025      
0.975]
-------------------------------------------------------------------------------
------
const               12.1101      0.178     67.915      0.000      11.760      
12.460
DIST_M              -0.0040   3.46e-05   -114.327      0.000      -0.004      
-0.004
PLACE               -0.0010      0.000     -4.299      0.000      -0.001      
-0.001
DATE_LABEL          -0.0108      0.007     -1.651      0.099      -0.024      
0.002
TEMP_F              -0.0077      0.001     -6.269      0.000      -0.010      
-0.005
GRADE               -0.0033      0.008     -0.421      0.674      -0.019      
0.012
TIME2                0.5459      0.003    181.963      0.000       0.540      
0.552
COURSE_TYPE_LABEL    0.1193      0.010     12.088      0.000       0.100      
0.139
==============================================================================
Omnibus:                      265.446   Durbin-Watson:                   0.260
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            10032.194
Skew:                           1.633   Prob(JB):                         0.00
Kurtosis:                      24.700   Cond. No.                     9.66e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 9.66e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [17]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
```

In [18]:
```
# Drops constant, split data into test and train sets
# Use 'PLACE' as a proxy for pace since it is not a classification
X_train, X_test, y_train, y_test = train_test_split(data.drop('PLACE', axis=1), d

# fitting the model using randomforestclassifier
model = RandomForestClassifier(n_estimators=500, n_jobs=-1, random_state=42)
model.fit(X_train, y_train)

# plotting feature importances
features = data.drop('PLACE', axis=1).columns
importances = model.feature_importances_
indices = np.argsort(importances)
plt.figure(figsize=(10,15))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Drop distance feature

In [19]:
```
# data[['PLACE','TEMP_F','GRADE','TIME2','PACE_CALC2','DATE_LABEL']]

# Date Labels: 0-5 (beginning week of season - end week of season)
# Temperature: Any temp in Fahrenheit
# Grade : 5-8
# Course Type: 0=FLAT,1=HILLY,2=ROLLING


# set dependent and independents
X = data[['DATE_LABEL','TEMP_F','GRADE','COURSE_TYPE_LABEL']]
y = data['PACE_CALC2']
```

Modeling to Predict Pace based on Temp, Grade, and Date, Course Type

In [20]:
```
regr = linear_model.LinearRegression()
regr.fit(X.values, y)
```

Out[20]:
```
LinearRegression()
```

In [21]:
```
# pair the feature names with the coefficients
list(zip(X, regr.coef_))
```

Out[21]:
```
[('DATE_LABEL', -0.3592647673389674),
 ('TEMP_F', 0.035480054305750894),
 ('GRADE', 0.0646512753170511),
 ('COURSE_TYPE_LABEL', 0.7130015911618681)]
```

The first two cells illustrate predicted paces for the beginning (0) of the season and end (5) of the season for an 8th grader. The third cell illustrates the impact changing course type has on the predicted pace, changing from flat to hilly slows the predicted pace.

In [22]:
```
# Enter independent values to predict pace
DATE = 0 # Date Labels: 0-5 (beginning of season through end of season)
TEMP = 65 # Temperature in F
GRADE = 8 # Grade 5-8
COURSETYPE = 0 # COURSE TYPE: 0=FLAT,1=HILLY,2=ROLLING

predictedpace = regr.predict([[DATE,TEMP,GRADE,COURSETYPE]])
print(predictedpace)
```

```
[9.41096864]
```

In [23]:
```
# Enter independent values to predict pace
DATE = 5 # Date Labels: 0-5 (beginning of season through end of season)
TEMP = 65 # Temperature in F
GRADE = 8 # Grade 5-8
COURSETYPE = 0 # COURSE TYPE: 0=FLAT,1=HILLY,2=ROLLING

predictedpace = regr.predict([[DATE,TEMP,GRADE,COURSETYPE]])
print(predictedpace)
```

```
[7.6146448]
```

In [24]:
```
# Enter independent values to predict pace
DATE = 5 # Date Labels: 0-5 (beginning of season through end of season)
TEMP = 65 # Temperature in F
GRADE = 8 # Grade 5-8
COURSETYPE = 1 # COURSE TYPE: 0=FLAT,1=HILLY,2=ROLLING

predictedpace = regr.predict([[DATE,TEMP,GRADE,COURSETYPE]])
print(predictedpace)
```

```
[8.32764639]
```

These next two cells illustrate predicted paces for the beginning (0) of the season and end (5) of the season for an 6th grader. Note the lower predicted paces using the same inputs as the 8th grade example above. The final cell shows the impact of increasing the temperature on the predicted pace.

In [25]:
```
# Enter independent values to predict pace
DATE = 0 # Date Labels: 0-5 (beginning of season through end of season)
TEMP = 65 # Temperature in F
GRADE = 6 # Grade 5-8
COURSETYPE = 0 # COURSE TYPE: 0=FLAT,1=HILLY,2=ROLLING

predictedpace = regr.predict([[DATE,TEMP,GRADE,COURSETYPE]])
print(predictedpace)
```

```
[9.28166609]
```

In [26]:
```
# Enter independent values to predict pace
DATE = 5 # Date Labels: 0-5 (beginning of season through end of season)
TEMP = 65 # Temperature in F
GRADE = 6 # Grade 5-8
COURSETYPE = 0 # COURSE TYPE: 0=FLAT,1=HILLY,2=ROLLING

predictedpace = regr.predict([[DATE,TEMP,GRADE,COURSETYPE]])
print(predictedpace)
```

```
[7.48534225]
```

In [27]:
```
# Enter independent values to predict pace
DATE = 5 # Date Labels: 0-5 (beginning of season through end of season)
TEMP = 80 # Temperature in F
GRADE = 6 # Grade 5-8
COURSETYPE = 0 # COURSE TYPE: 0=FLAT,1=HILLY,2=ROLLING

predictedpace = regr.predict([[DATE,TEMP,GRADE,COURSETYPE]])
print(predictedpace)
```

```
[8.01754307]
```

Follow-up review challenges and limitations of data, better performance of lower/younger grades

In [28]:
```
## Mean of the top 10 from each race is closer to 6th grade than 8th
df = data.loc[data['PLACE'] <= 10]
df['GRADE'].mean()
```

Out[28]:
```
6.933333333333334
```

In [29]:
```
data.loc[data['PLACE'] <= 10].mean()
```

Out[29]:
```
DIST_M               2933.333333
PLACE                   5.500000
DATE_LABEL              2.500000
TEMP_F                 69.000000
GRADE                   6.933333
TIME2                  13.878611
PACE_CALC2              7.588611
COURSE_TYPE_LABEL       1.000000
dtype: float64
```

In [30]:
```
data.describe()
```

Out[30]:

	DIST_M	PLACE	DATE_LABEL	TEMP_F	GRADE	TIME2	PACE_CALC2	C
count	500.000000	500.00000	500.000000	500.000000	500.000000	500.000000	500.000000	
mean	2957.200000	58.99800	2.738000	71.444000	7.184000	17.297567	9.319033	
std	194.122454	46.11337	1.883734	8.658915	0.720543	3.291983	1.985757	
min	2400.000000	1.00000	0.000000	58.000000	5.000000	11.566667	6.083333	
25%	3000.000000	21.00000	1.000000	64.000000	7.000000	15.016667	7.912500	
50%	3000.000000	42.00000	2.500000	67.000000	7.000000	16.633333	8.833333	
75%	3000.000000	96.00000	5.000000	82.000000	8.000000	18.716667	10.216667	
max	3200.000000	164.00000	5.000000	82.000000	8.000000	30.983333	17.583333	

In [31]:
```
eighth_pace = data.loc[data['GRADE'] == 8]
eighth_pace['PACE_CALC2'].describe()
```

Out[31]:
```
count    180.000000
mean       9.335833
std        1.925735
min        6.683333
25%        7.979167
50%        8.900000
75%       10.266667
max       17.583333
Name: PACE_CALC2, dtype: float64
```

In [32]:
```
seventh_pace = data.loc[data['GRADE'] == 7]
seventh_pace['PACE_CALC2'].describe()
```

Out[32]:
```
count    236.000000
mean       9.693362
std        2.103619
min        6.083333
25%        8.095833
50%        9.141667
75%       10.750000
max       16.750000
Name: PACE_CALC2, dtype: float64
```

In [33]:
```
sixth_pace = data.loc[data['GRADE'] == 6]
sixth_pace['PACE_CALC2'].describe()
```

Out[33]:
```
count    80.000000
mean      8.260625
std       1.273563
min       6.533333
25%       7.479167
50%       7.983333
75%       8.683333
max      12.833333
Name: PACE_CALC2, dtype: float64
```

New model w/o Grade

In [34]:
```
X2 = X[['DATE_LABEL','TEMP_F','COURSE_TYPE_LABEL']]
y2 = y
```

In [35]:
```
regr = linear_model.LinearRegression()
regr.fit(X2.values, y2)
```

Out[35]:
```
LinearRegression()
```

In [36]:
```
# pair the feature names with the coefficients
list(zip(X2, regr.coef_))
```

Out[36]:
```
[('DATE_LABEL', -0.35945962485477884),
 ('TEMP_F', 0.0366494707448249),
 ('COURSE_TYPE_LABEL', 0.716794357837025)]
```

In [37]:
```
# Enter independent values to predict pace
DATE = 1 # Date Labels: 0-5 (beginning of season through end of season)
TEMP = 70 # Temperature in F
COURSETYPE = 2 # COURSE TYPE: 0=FLAT,1=HILLY,2=ROLLING

predictedpace = regr.predict([[DATE,TEMP,COURSETYPE]])
print(predictedpace)
```

```
[10.60477951]
```

In [38]:
```
# with statsmodels
X2 = sm.add_constant(X2) # adding a constant
```

In [39]:
```
# fit & predict model
model = sm.OLS(y, X2).fit()
predictions = model.predict(X2)
```

In [40]:
```
#set model summary as variable to print
print_model = model.summary()
print(print_model)
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:             PACE_CALC2   R-squared:                       0.322
Model:                            OLS   Adj. R-squared:                  0.317
Method:                 Least Squares   F-statistic:                     78.37
Date:                Fri, 01 Jul 2022   Prob (F-statistic):           1.65e-41
Time:                        10:38:06   Log-Likelihood:                -954.97
No. Observations:                 500   AIC:                             1918.
Df Residuals:                     496   BIC:                             1935.
Df Model:                           3                                         
Covariance Type:            nonrobust                                         
================================================================================
======
                        coef    std err          t      P>|t|      [0.025      
0.975]
--------------------------------------------------------------------------------
------
const                 6.9652      0.998      6.977      0.000       5.004       
8.927
DATE_LABEL           -0.3595      0.057     -6.330      0.000      -0.471      
-0.248
TEMP_F                0.0366      0.012      2.992      0.003       0.013       
0.061
COURSE_TYPE_LABEL     0.7168      0.125      5.725      0.000       0.471       
0.963
==============================================================================
Omnibus:                       92.910   Durbin-Watson:                   0.328
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              168.075
Skew:                           1.073   Prob(JB):                     3.18e-37
Kurtosis:                       4.861   Cond. No.                         981.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
```