big-data-derby

November 13, 2022

#

Big Data Derby - 2022

The goal of this competition is to analyze horse racing tactics, drafting strategies, and path efficiency. You will develop a model using never-before-released coordinate data along with basic race information.

Your work will help racing horse owners, trainers, and veterinarians better understand how equine performance and welfare fit together. With better data analysis, equine welfare could significantly improve.

0.1 Context

Injury prevention is a critical component in modern athletics. Sports that involve animals, such as horse racing, are no different than human sport. Typically, efficiency in movement correlates to both improvements in performance and injury prevention.

A wealth of data is now collected, including measures for heart rate, EKG, longitudinal movement, dorsal/ventral movement, medial/lateral deviation, total power and total landing vibration. Your data science skills and analysis are needed to decipher what makes the most positive impact.

In this competition, you will create a model to interpret one aspect of this new data. You'll be among the first to access X/Y coordinate mapping of horses during races. Using the data, you might analyze jockey decision making, compare race surfaces, or measure the relative importance of drafting. With considerable data, contestants can flex their creativity problem solving skills.

The New York Racing Association (NYRA) and the New York Thoroughbred Horsemen's Association (NYTHA) conduct world class thoroughbred racing at Aqueduct Racetrack, Belmont Park and Saratoga Race Course.

With your help, NYRA and NYTHA will better understand their vast data set, which could lead to new ways of racing and training in a highly traditional industry. With improved use of horse tracking data, you could help improve equine welfare, performance and rider decision making.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import os
import gc # garbage collector
```

```
plt.style.use("fivethirtyeight")
sns.set_context('paper', font_scale=1.6)

%matplotlib inline
%reload_ext autoreload
%autoreload 2
```

[2]:
```
os.listdir('./data')
```

[2]:
```
['nyra_start_table.csv',
 'nyra_race_table.csv',
 'nyra_2019_complete.csv',
 'nyra_tracking_table.csv']
```

[3]:
```
%%time
start_df = pd.read_csv('./data/nyra_start_table.csv')
race_df = pd.read_csv('./data/nyra_race_table.csv')
track_df = pd.read_csv('./data/nyra_tracking_table.csv')
complete_df = pd.read_csv('./data/nyra_2019_complete.csv', low_memory=False)
```

```
CPU times: user 7.84 s, sys: 1.84 s, total: 9.68 s
Wall time: 9.86 s
```

[4]:
```
complete_df.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5228430 entries, 0 to 5228429
Data columns (total 17 columns):
 #   Column           Dtype
---  ------           -----
 0   track_id         object
 1   race_date        object
 2   race_number      int64
 3   program_number   object
 4   trakus_index     int64
 5   latitude         float64
 6   longitude        float64
 7   distance_id      int64
 8   course_type      object
 9   track_condition  object
 10  run_up_distance  int64
 11  race_type        object
 12  purse            int64
 13  post_time        int64
 14  weight_carried   int64
 15  jockey           object
 16  odds             int64
```

```
dtypes: float64(2), int64(8), object(7)
memory usage: 2.5 GB
```

0.2 Downsample Data

0.2.1 Integer Columns

[5]:
```python
def downcast_df_int_columns(df):
    list_of_columns = list(df.select_dtypes(include=["int32", "int64"]).columns)

    if len(list_of_columns)>=1:
        max_string_length = max([len(col) for col in list_of_columns]) # finds␣
↪max string length for better status printing
        print("downcasting integers for:", list_of_columns, "\n")

        for col in list_of_columns:
            print("reduced memory usage for:  ", col.
↪ljust(max_string_length+2)[:max_string_length+2],
                  "from", str(round(df[col].memory_usage(deep=True)*1e-6,2)).
↪rjust(8), "to", end=" ")
            df[col] = pd.to_numeric(df[col], downcast="integer")
            print(str(round(df[col].memory_usage(deep=True)*1e-6,2)).rjust(8))
    else:
        print("no columns to downcast")

    gc.collect()

    print("done")

downcast_df_int_columns(complete_df)
downcast_df_int_columns(race_df)
downcast_df_int_columns(track_df)
downcast_df_int_columns(start_df)
```

```
downcasting integers for: ['race_number', 'trakus_index', 'distance_id',
'run_up_distance', 'purse', 'post_time', 'weight_carried', 'odds']

reduced memory usage for:   race_number       from    41.83 to     5.23
reduced memory usage for:   trakus_index      from    41.83 to    10.46
reduced memory usage for:   distance_id       from    41.83 to    10.46
reduced memory usage for:   run_up_distance   from    41.83 to    10.46
reduced memory usage for:   purse             from    41.83 to    20.91
reduced memory usage for:   post_time         from    41.83 to    10.46
reduced memory usage for:   weight_carried    from    41.83 to    10.46
reduced memory usage for:   odds              from    41.83 to    10.46
done
downcasting integers for: ['race_number', 'distance_id', 'run_up_distance',
'purse', 'post_time']
```

```
reduced memory usage for:   race_number       from     0.02 to      0.0
reduced memory usage for:   distance_id       from     0.02 to      0.0
reduced memory usage for:   run_up_distance   from     0.02 to      0.0
reduced memory usage for:   purse             from     0.02 to     0.01
reduced memory usage for:   post_time         from     0.02 to      0.0
done
downcasting integers for: ['race_number', 'trakus_index']

reduced memory usage for:   race_number    from    41.83 to     5.23
reduced memory usage for:   trakus_index   from    41.83 to    10.46
done
downcasting integers for: ['race_number', 'weight_carried', 'odds']

reduced memory usage for:   race_number      from     0.12 to     0.02
reduced memory usage for:   weight_carried   from     0.12 to     0.03
reduced memory usage for:   odds             from     0.12 to     0.03
done
```

0.2.2 Float Columns

[6]:
```
def downcast_df_float_columns(df):
    list_of_columns = list(df.select_dtypes(
        include=[float, np.float16, np.float32, np.float64, np.float128]).
 ↪columns)

    if len(list_of_columns)>=1:
        max_string_length = max([len(col) for col in list_of_columns]) # finds␣
 ↪max string length for better status printing
        print("downcasting float for:", list_of_columns, "\n")

        for col in list_of_columns:
            print("reduced memory usage for:  ", col.
 ↪ljust(max_string_length+2)[:max_string_length+2],
                  "from", str(round(df[col].memory_usage(deep=True)*1e-6,2)).
 ↪rjust(8), "to", end=" ")
            df[col] = pd.to_numeric(df[col], downcast="float")
            print(str(round(df[col].memory_usage(deep=True)*1e-6,2)).rjust(8))
    else:
        print("no columns to downcast")

    gc.collect()

    print("done")

downcast_df_float_columns(complete_df)
downcast_df_float_columns(race_df)
```

```
downcast_df_float_columns(track_df)
downcast_df_float_columns(start_df)
```

```
downcasting float for: ['latitude', 'longitude']

reduced memory usage for:   latitude    from    41.83 to    20.91
reduced memory usage for:   longitude   from    41.83 to    20.91
done
no columns to downcast
done
downcasting float for: ['latitude', 'longitude']

reduced memory usage for:   latitude    from    41.83 to    20.91
reduced memory usage for:   longitude   from    41.83 to    20.91
done
no columns to downcast
done
```

0.2.3 Object Type

[7]:
```
def convert_columns_to_catg(df, column_list):
    for col in column_list:
        print("converting", col.ljust(30), "size: ", round(df[col].
 ↪memory_usage(deep=True)*1e-6,2), end="\t")
        df[col] = df[col].astype("category")
        print("->\t", round(df[col].memory_usage(deep=True)*1e-6,2))
        df[col] = df[col].apply(lambda x: x.strip())

convert_columns_to_catg(complete_df, complete_df.
 ↪select_dtypes(include="object").columns.to_list())
convert_columns_to_catg(track_df, track_df.select_dtypes(include="object").
 ↪columns.to_list())
convert_columns_to_catg(race_df, race_df.select_dtypes(include="object").
 ↪columns.to_list())
convert_columns_to_catg(start_df, start_df.select_dtypes(include="object").
 ↪columns.to_list())
```

```
converting track_id                         size:  313.71 ->       5.23
converting race_date                        size:  350.3  ->       10.48
converting program_number                   size:  313.71 ->       5.23
converting course_type                      size:  303.25 ->       5.23
converting track_condition                  size:  313.71 ->       5.23
converting race_type                        size:  313.71 ->       5.23
converting jockey                           size:  371.75 ->       10.47
converting track_id                         size:  313.71 ->       5.23
converting race_date                        size:  350.3  ->       10.48
converting program_number                   size:  313.71 ->       5.23
converting track_id                         size:  0.12   ->       0.0
```

```
converting race_date                          size:  0.13   ->       0.03
converting course_type                        size:  0.12   ->       0.0
converting track_condition                    size:  0.12   ->       0.0
converting race_type                          size:  0.12   ->       0.0
converting track_id                           size:  0.9    ->       0.02
converting race_date                          size:  1.0    ->       0.05
converting program_number                     size:  0.9    ->       0.02
converting jockey                             size:  1.06   ->       0.05
```

0.3 Data

0.3.1 File descriptions

- `nyra_start_table.csv` - horse/jockey race data
- `nyra_race_table.csv` - racetrack race data
- `nyra_tracking_table.csv` - tracking data
- `nyra_2019_complete.csv` - combined table of three above files

0.3.2 Columns

nyra_start_table.csv

- `track_id` - 3 character id for the track the race took place at.
  - `AQU` -Aqueduct
  - `BEL` - Belmont
  - `SAR` - Saratoga
- `race_date` - date the race took place. YYYY-MM-DD.
- `race_number` - Number of the race. Passed as 3 characters but can be cast or converted to int for this data set.
- `program_number` - Program number of the horse in the race passed as 3 characters. Should remain 3 characters as it isn't limited to just numbers. Is essentially the unique identifier of the horse in the race.
- `weight_carried` - An integer of the weight carried by the horse in the race.
- `jockey` - Name of the jockey on the horse in the race. 50 character max.
- `odds` - Odds to win the race passed as an integer. Divide by 100 to derive the odds to 1.
  - Example - 1280 would be 12.8-1.

nyra_race_table.csv

- `track_id` - 3 character id for the track the race took place at.
  - `AQU` -Aqueduct
  - `BEL` - Belmont
  - `SAR` - Saratoga
- `race_date` - date the race took place. YYYY-MM-DD.
- `race_number` - Number of the race. Passed as 3 characters but can be cast or converted to int for this data set.
- `distance_id` - Distance of the race in furlongs passed as an integer. Example - 600 would be 6 furlongs.
- `course_type` - The course the race was run over passed as one character.
  - `M` - Hurdle

  - D - Dirt
  - O - Outer turf
  - I - Inner turf
  - T - turf.
- track_condition - The condition of the course the race was run on passed as three characters.
  - YL - Yielding
  - FM - Firm
  - SY - Sloppy
  - GD - Good
  - FT - Fast
  - MY - Muddy
  - SF - Soft.
- run_up_distance - Distance in feet of the gate to the start of the race passed as an integer.
- race_type - The classification of the race passed as as five characters.
  - STK - Stakes
  - WCL - Waiver Claiming
  - WMC - Waiver Maiden Claiming
  - SST - Starter Stakes
  - SHP - Starter Handicap
  - CLM - Claiming
  - STR - Starter Allowance
  - AOC - Allowance Optionl Claimer
  - SOC - Starter Optional Claimer
  - MCL - Maiden Claiming
  - ALW - Allowance
  - MSW - Maiden Special Weight.
- purse - Purse in US dollars of the race passed as an money with two decimal places.
- post_time - Time of day the race began passed as 5 character. Example - 01220 would be 12:20.

nyra_tracking_table.csv

- track_id - 3 character id for the track the race took place at.
  - AQU -Aqueduct
  - BEL - Belmont
  - SAR - Saratoga
- race_date - date the race took place. YYYY-MM-DD.
- race_number - Number of the race. Passed as 3 characters but can be cast or converted to int for this data set.
- program_number - Program number of the horse in the race passed as 3 characters. Should remain 3 characters as it isn't limited to just numbers. Is essentially the unique identifier of the horse in the race.
- trakus_index - The common collection of point of the lat / long of the horse in the race passed as an integer. From what we can tell, it's collected every 0.25 seconds.
- latitude - The latitude of the horse in the race passed as a float.
- longitude - The longitude of the horse in the race passed as a float.

nyra_2019_complete.csv

- This file is the combined 3 files into one table. The keys to join them trakus with race - track_id, race_date, race_number. To join trakus with start - track_id, race_date, race_number, program_number.
- Data:
  - `track_id` - char(3)
  - `race_date` - date
  - `race_number` - char(3)
  - `program_number` - char(3)
  - `trakus_index` - int
  - `latitude` - float
  - `longitude` - float
  - `distance_id` - int
  - `course_type` - char(1)
  - `track_condition` - char(3)
  - `run_up_distance` - int
  - `race_type` - char(5)
  - `post_time` - char(5)
  - `weight_carried` - int
  - `jockey` - char(50)
  - `odds` - int

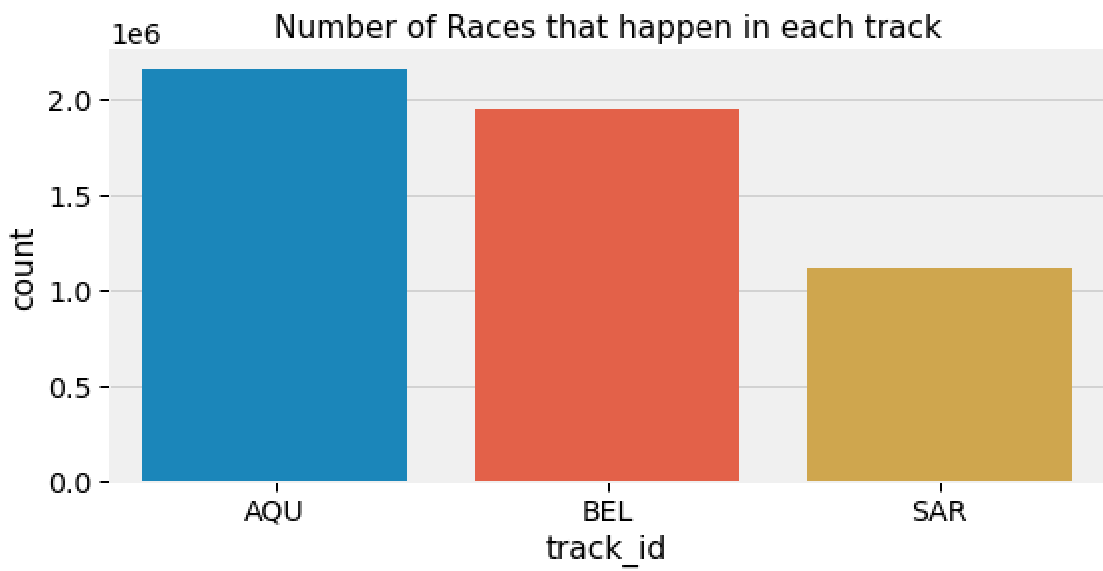
0.4 EDA

0.4.1 Let's convert the timeseries variable to a datetime object of pandas.

[8]:
```
complete_df['race_date'] = pd.to_datetime(complete_df['race_date'])
start_df['race_date'] = pd.to_datetime(start_df['race_date'])
track_df['race_date'] = pd.to_datetime(start_df['race_date'])
race_df['race_date'] = pd.to_datetime(race_df['race_date'])
```

0.4.2 How many races happen on each track through out the event?

[9]:
```
# px.histogram(data_frame=complete_df, x='track_id', title='Number of Races␣
↪that happen in each track')

plt.figure(figsize=(8, 4))
sns.countplot(x='track_id', data=complete_df)
plt.title('Number of Races that happen in each track')
plt.show()
```



0.4.3 How many races have each of the jockeys participated?

[10]:
```
px.histogram(data_frame=complete_df,
             x='jockey',
             title='Number of Races participated by each of the Jockey',
            ).update_xaxes(categoryorder='total descending')
```

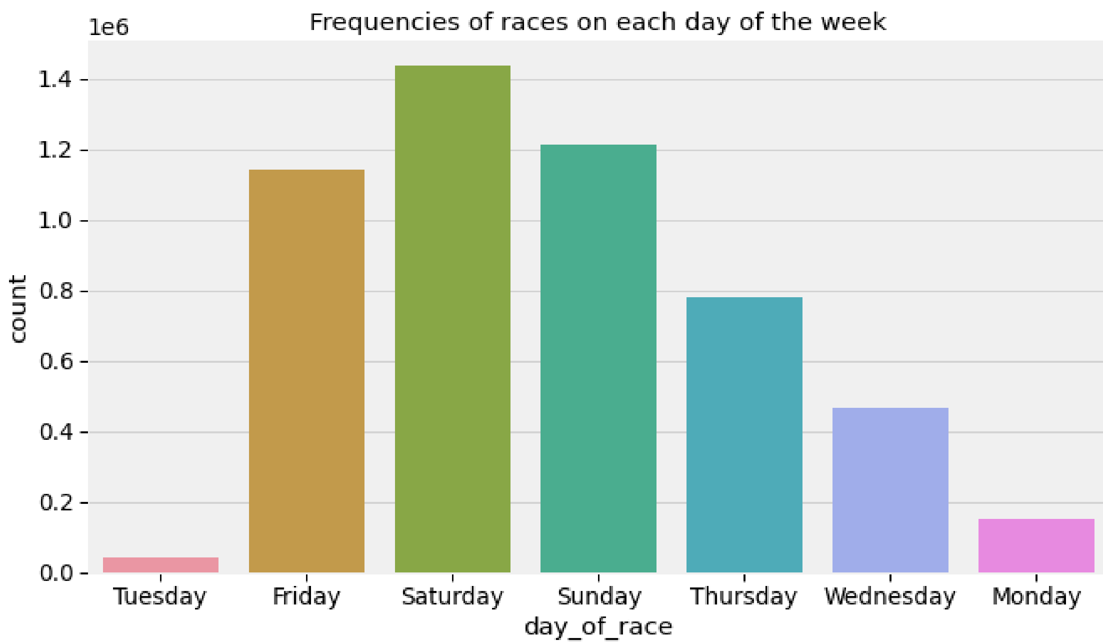
0.4.4 Now, let's see how many races occur on different days.

[11]:
```
complete_df['day_of_race'] = complete_df['race_date'].dt.day_name()
complete_df.sample(3)
```

[11]:
```
        track_id  race_date  race_number program_number  trakus_index  \
2916826      AQU 2019-11-16           10              3            86
2568546      AQU 2019-04-05            1              1            11
1388860      AQU 2019-01-05            4             1A           158

          latitude  longitude  distance_id course_type track_condition  \
2916826  40.673328 -73.827812          600           O              FM
2568546  40.666931 -73.830330          800           D              FT
1388860  40.675488 -73.828117          650           D              SY

         run_up_distance race_type  purse  post_time  weight_carried  \
2916826               45       CLM  38000        423             122
2568546               54       SOC  60000        130             120
1388860               32       MSW  60000        158             121

                     jockey  odds day_of_race
```

```
2916826       Irad Ortiz Jr.  1090     Saturday
2568546  Kendrick Carmouche  1830       Friday
1388860         Dylan Davis   540     Saturday
```

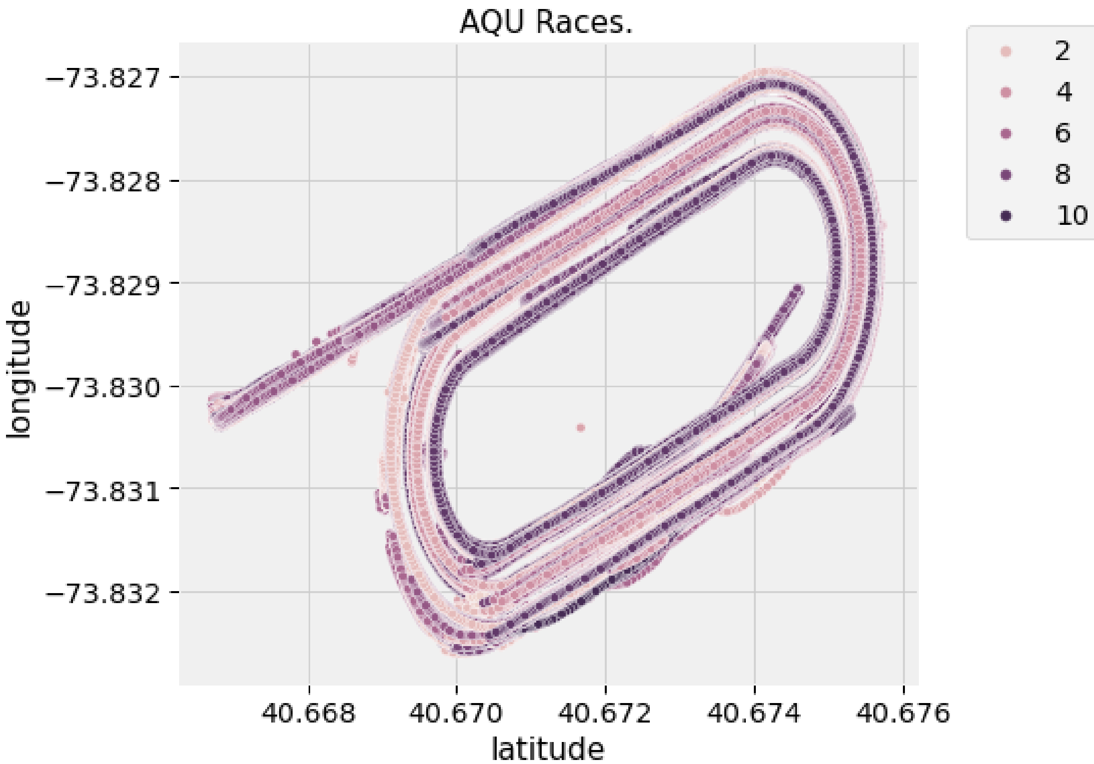
[12]:
```
# px.histogram(data_frame=complete_df, x='day_of_race',
#               title='Frequencies of races on each day of the week'
#              ).update_xaxes(categoryorder='total descending')

plt.figure(figsize=(10, 6), dpi=70)
sns.countplot(x='day_of_race', data=complete_df)
plt.title('Frequencies of races on each day of the week')
plt.show()
```



0.4.5 Let's look at the tracks itself.

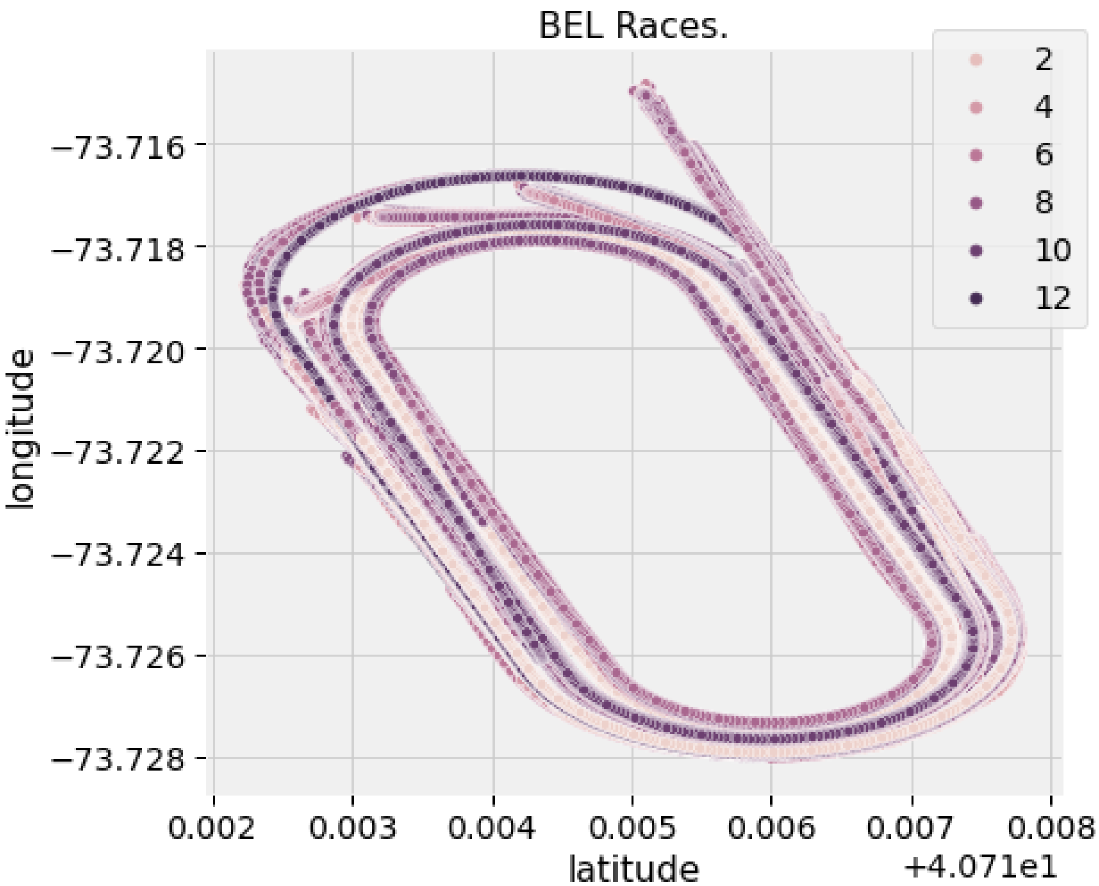
[13]:
```
AQU = complete_df[complete_df['track_id'] == 'AQU']
BEL = complete_df[complete_df['track_id'] == 'BEL']
SAR = complete_df[complete_df['track_id'] == 'SAR']
```

[14]:
```
plt.figure(figsize=(6, 6))
sns.scatterplot(data=AQU, x='latitude', y='longitude', hue='race_number')
plt.title('AQU Races.')
plt.legend(bbox_to_anchor=(1.05, 1.05));
```

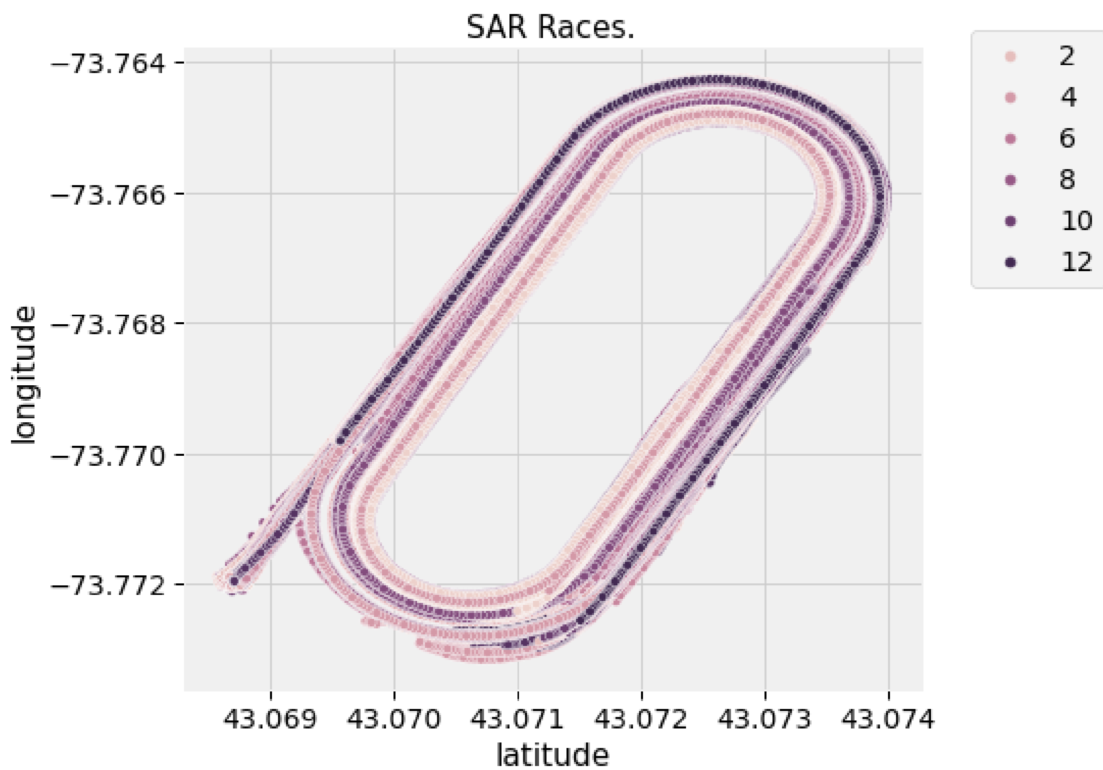


[15]:

```
plt.figure(figsize=(6, 6))
sns.scatterplot(data=BEL, x='latitude', y='longitude', hue='race_number')
plt.title('BEL Races.')
plt.legend(bbox_to_anchor=(1.05, 1.05));
```



[16]:
```
plt.figure(figsize=(6, 6))
sns.scatterplot(data=SAR, x='latitude', y='longitude', hue='race_number')
plt.title('SAR Races.')
plt.legend(bbox_to_anchor=(1.05, 1.05));
```



0.4.6 How much does race type affect the Purse?

- Purse is essentially the bet the person has had.

[17]:
```
complete_df.groupby('race_type').mean()
```

[17]:
```
           race_number  trakus_index   latitude  longitude  distance_id  \
race_type
ALW           6.093538    185.846467  41.282330 -73.774216   770.302899
AOC           5.916968    184.032137  41.243813 -73.775284   774.279038
CLM           4.617937    176.261663  41.084877 -73.783241   726.692836
MCL           5.389970    175.746179  40.990719 -73.781471   705.253299
MSW           4.748384    177.241665  41.316696 -73.772995   723.229631
SHP           2.790013    215.753521  40.672249 -73.829826   889.500640
SOC           5.132080    164.895866  40.905907 -73.805199   675.197538
SST           6.265131    192.360377  40.672638 -73.829636   773.378084
STK           7.517272    216.806986  41.416714 -73.769341   902.099022
STR           5.328535    178.313027  41.297817 -73.770485   740.485370
WCL           3.952649    164.424685  41.368858 -73.774506   687.541320
WMC          10.000000    178.000000  40.715492 -73.724045   700.000000

           run_up_distance         purse   post_time  weight_carried  \
race_type
```

```
ALW              67.073798   73746.063113  444.211875       122.309352
AOC              75.511760   78967.252923  394.677242       121.428105
CLM              61.714695   41092.578673  387.924718       120.987450
MCL              65.106957   39693.855428  429.738993       119.623017
MSW              72.870640   72970.160845  422.242846       119.175872
SHP              52.806658   47983.354673  613.185659       120.427657
SOC              49.683690   58320.164130  323.864316       120.473731
SST              42.660595   63667.815675  419.940747       121.717271
STK              70.315405  280162.767394  478.781444       122.020057
STR              68.753422   56499.941096  375.907804       121.632866
WCL              69.917627   47715.357813  241.438399       120.635710
WMC              90.000000   41000.000000  512.000000       121.250000

                  odds
race_type
ALW        1462.921695
AOC        1025.157295
CLM        1347.592074
MCL        1953.557041
MSW        1604.390126
SHP        1292.129748
SOC        1063.948682
SST        1542.774492
STK        1377.464710
STR        1111.181457
WCL        1400.884481
WMC        3784.583333
```

[18]:

```
complete_df.groupby('race_type').mean()['purse']
```

[18]:

```
race_type
ALW     73746.063113
AOC     78967.252923
CLM     41092.578673
MCL     39693.855428
MSW     72970.160845
SHP     47983.354673
SOC     58320.164130
SST     63667.815675
STK    280162.767394
STR     56499.941096
WCL     47715.357813
WMC     41000.000000
Name: purse, dtype: float64
```

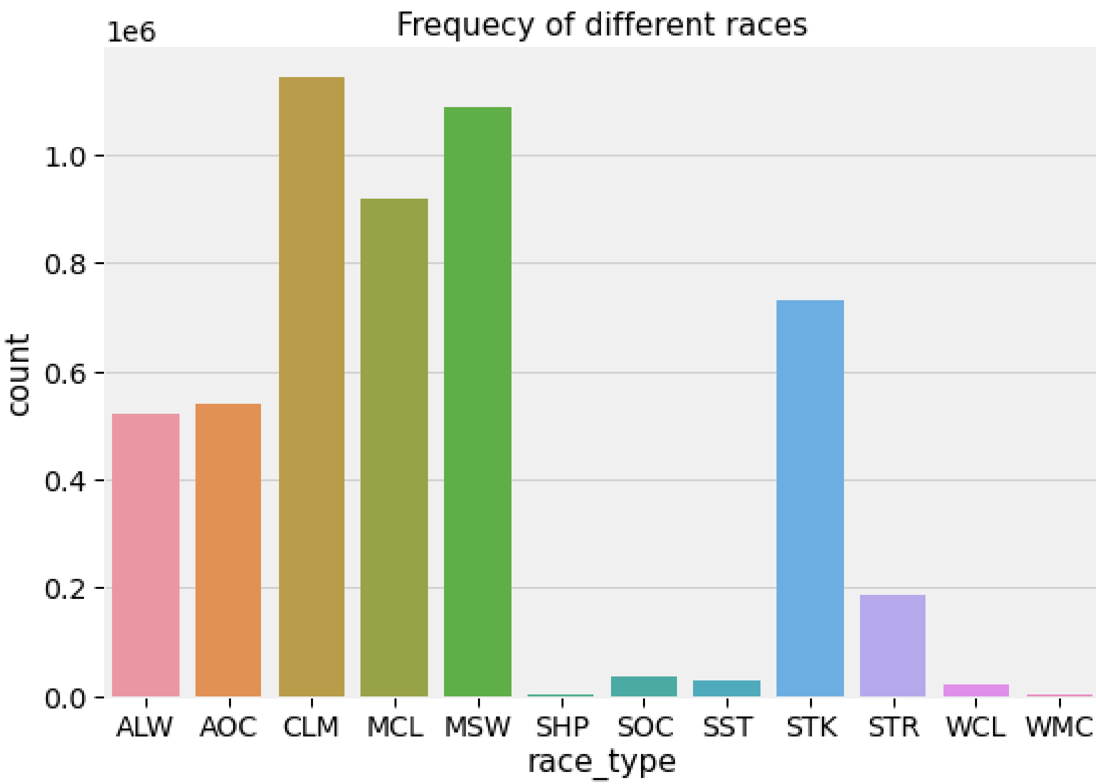
[19]:

```
px.bar(data_frame=complete_df.groupby('race_type').mean(), x='purse')
# plt.figure(figsize=(6, 6))
```

```
# sns.histplot(data=complete_df.groupby('race_type').mean(), x='purse',␣
↪hue=complete_df.groupby('race_type').mean().index.to_list())
# plt.title('Average Purse per Type')
# plt.legend(bbox_to_anchor=(1.05, 1.05));
```

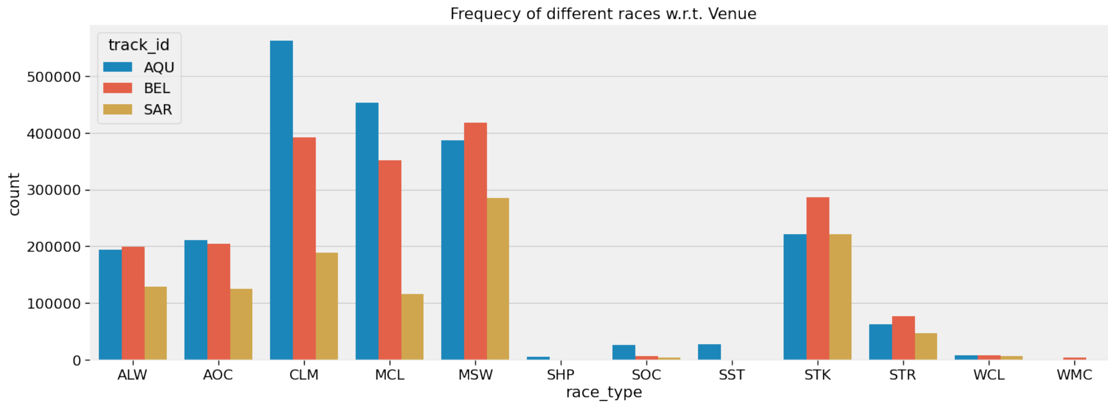
0.4.7 What are the types of race_types?

[20]:
```
plt.figure(figsize=(8, 6))
sns.countplot(x='race_type', data=complete_df)
plt.title('Frequecy of different races')
plt.show()
```



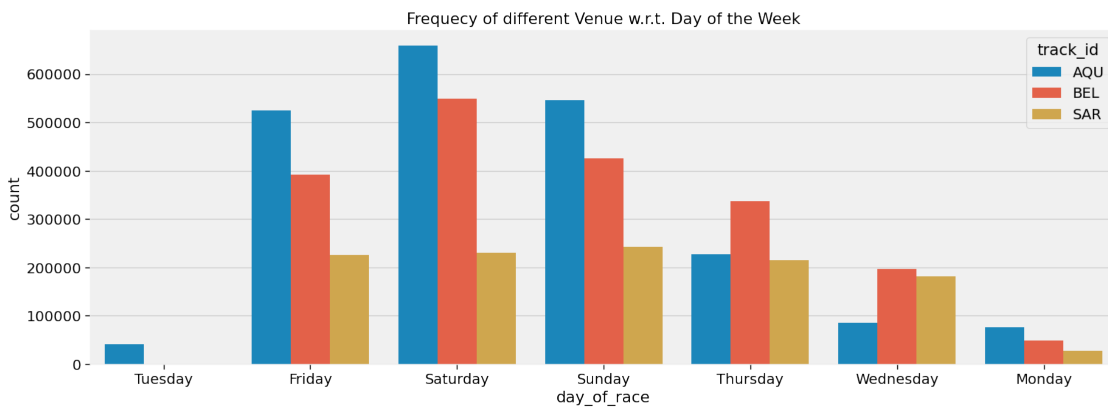
0.4.8 How many of the Races in different Venues?

[21]:
```
plt.figure(figsize=(16, 6), dpi=100)
sns.countplot(x='race_type', data=complete_df, hue='track_id')
plt.title('Frequecy of different races w.r.t. Venue')
plt.show()
```



0.4.9 How many races happen at differnent venues?

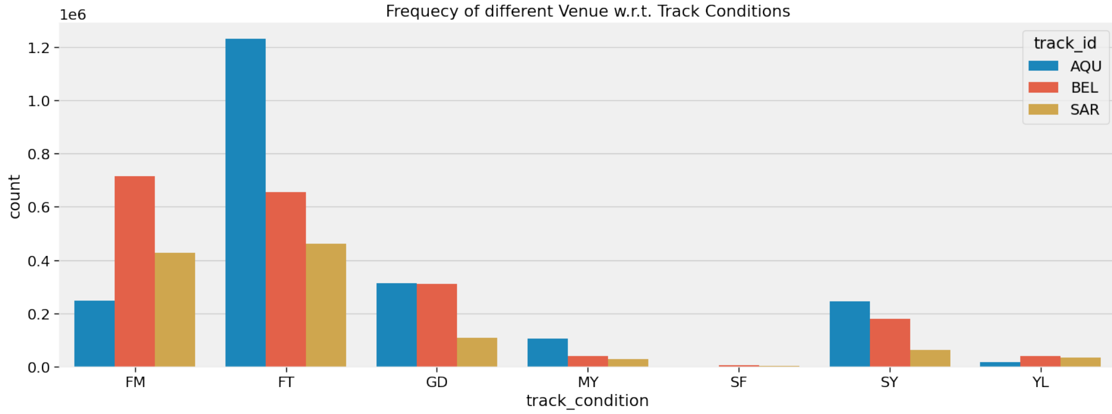
[22]:
```
plt.figure(figsize=(16, 6), dpi=100)
sns.countplot(x='day_of_race', data=complete_df, hue='track_id')
plt.title('Frequecy of different Venue w.r.t. Day of the Week')
plt.show()
```



0.4.10 How many of different track conditions at different venues?

[23]:
```
# px.histogram(data_frame=complete_df,
#              x='track_condition',
#              color='track_id',
#              title='Frequency of different Races.',
#             ).update_xaxes(categoryorder='total descending')

plt.figure(figsize=(16, 6), dpi=100)
sns.countplot(x='track_condition', data=complete_df, hue='track_id')
plt.title('Frequecy of different Venue w.r.t. Track Conditions')
```

```
plt.show()
```



0.5 Modeling

[24]:
```
complete_df.sample(5)
```

[24]:
```
        track_id  race_date  race_number program_number  trakus_index  \
582092       BEL 2019-10-20            7              2           245
1618765      AQU 2019-11-15            7             1A            67
3881308      AQU 2019-11-07            8             12            13
3811768      AQU 2019-03-16            7              5           169
4221059      AQU 2019-11-16            7              1            40

          latitude  longitude  distance_id course_type track_condition  \
582092   40.715595 -73.727692          800           T              FM
1618765  40.672268 -73.828224          600           O              FM
3881308  40.669891 -73.829361          600           O              FM
3811768  40.675274 -73.829987          600           D              FT
4221059  40.670696 -73.828644          650           D              FT

         run_up_distance race_type  purse  post_time  weight_carried  \
582092               178       CLM  55000        341             124
1618765               45       CLM  48000        256             120
3881308               56       ALW  75000        348             118
3811768               45       CLM  50000        505             119
4221059               54       ALW  66000        255             119

                          jockey  odds day_of_race
582092            Irad Ortiz Jr.  1340      Sunday
1618765  Luis A. Rodriguez Castro   610      Friday
3881308              Eric Cancel  1160    Thursday
3811768          Michael J. Luzzi  2280    Saturday
```

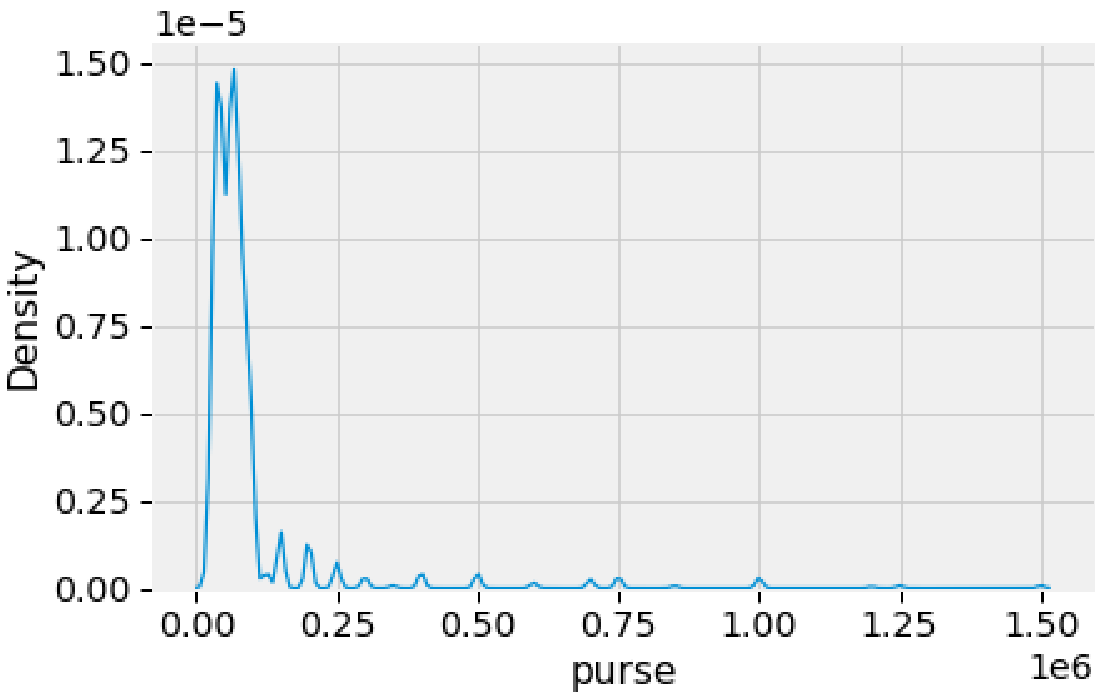
```
4221059        Samuel Camacho Jr.  2675    Saturday
```

[25]:
```
complete_df['year'] = complete_df['race_date'].dt.year
complete_df['month'] = complete_df['race_date'].dt.month
complete_df['day'] = complete_df['race_date'].dt.day
complete_df.drop(['race_date'], axis=1, inplace=True)
```

[26]:
```
complete_df.sample(5)
```

[26]:
```
        track_id  race_number program_number  trakus_index   latitude  \
5210900      BEL            1              7           362  40.714935
658850       BEL            9              6           127  40.717422
654367       BEL            5              8           118  40.717419
3496942      BEL            3              1           257  40.713863
4620739      AQU            9              4           219  40.673981

         longitude  distance_id course_type track_condition  run_up_distance  \
5210900 -73.726814         1800           M              FM                0
658850  -73.725250          700           T              YL              126
654367  -73.726936          650           D              SY               48
3496942 -73.722504          600           I              FM              126
4620739 -73.827858          800           T              FM               28

        race_type   purse  post_time  weight_carried            jockey  odds  \
5210900       STK   75000        302             146  Michael Mitchell  1700
658850        STK  125000        551             116     Manuel Franco   120
654367        MCL   35000        523             124    Irad Ortiz Jr.    85
3496942       MCL   41000        205             121       Dylan Davis  1360
4620739       MCL   46000        421             119     Luis R. Reyes  7025

        day_of_race  year  month  day
5210900   Wednesday  2019      9   18
658850     Saturday  2019      4   27
654367     Thursday  2019      5   23
3496942      Sunday  2019      9    8
4620739    Thursday  2019     11    7
```

[27]:
```
sns.kdeplot(x='purse', data=complete_df);
```



[54]:
```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

[29]:
```
X = complete_df.drop('purse', axis=1)
y = complete_df['purse']

categorical_cols = X.select_dtypes(exclude=np.number).columns.to_list()
numerical_cols = X.select_dtypes(np.number).columns.to_list()

categorical_cols, numerical_cols
```

[29]:
```
(['track_id',
  'program_number',
  'course_type',
  'track_condition',
  'race_type',
  'jockey',
  'day_of_race'],
 ['race_number',
  'trakus_index',
  'latitude',
  'longitude',
```

```
'distance_id',
'run_up_distance',
'post_time',
'weight_carried',
'odds',
'year',
'month',
'day'])
```

[30]:
```
encoders = {}

for col in categorical_cols:
    encoder = LabelEncoder()
    encoder.fit(complete_df[col])
    encoders[col] = encoder
```

[31]:
```
encoders
```

[31]:
```
{'track_id': LabelEncoder(),
 'program_number': LabelEncoder(),
 'course_type': LabelEncoder(),
 'track_condition': LabelEncoder(),
 'race_type': LabelEncoder(),
 'jockey': LabelEncoder(),
 'day_of_race': LabelEncoder()}
```

[32]:
```
import pickle
```

[33]:
```
with open('label_encoders.pkl', 'wb') as f:
    pickle.dump(encoders, f)
```

[34]:
```
with open('label_encoders.pkl', 'rb') as f:
    loaded_dict = pickle.load(f)
loaded_dict
```

[34]:
```
{'track_id': LabelEncoder(),
 'program_number': LabelEncoder(),
 'course_type': LabelEncoder(),
 'track_condition': LabelEncoder(),
 'race_type': LabelEncoder(),
 'jockey': LabelEncoder(),
 'day_of_race': LabelEncoder()}
```

[35]:
```
kfold = KFold(n_splits=10)
```

[55]:
```
def evaluate(x_train, x_test, y_train, y_test, model):
    x_train_pred = model.predict(x_train)
    x_test_pred = model.predict(x_test)
```

```
    print(f'''
Train Predictions:
    * MSE: {mean_squared_error(y_true=y_train, y_pred=x_train_pred):.2f}
    * RMSE: {np.sqrt(mean_squared_error(y_true=y_train, y_pred=x_train_pred)):.
↪2f}
    * MAE: {mean_absolute_error(y_true=y_train, y_pred=x_train_pred):.2f}
    * r2: {r2_score(y_true=y_train, y_pred=x_train_pred)}

Test Predictions:
    * MSE: {mean_squared_error(y_true=y_test, y_pred=x_test_pred):.2f}
    * RMSE: {np.sqrt(mean_squared_error(y_true=y_test, y_pred=x_test_pred)):.2f}
    * MAE: {mean_absolute_error(y_true=y_test, y_pred=x_test_pred):.2f}
    * r2: {r2_score(y_true=y_test, y_pred=x_test_pred)}
-----------------------------------------------------------
''')
```

[37]:
```
for col, encoder in encoders.items():
    X[col] = encoder.transform(X[col])
```

[56]:
```
for ix, (train_ix, test_ix) in enumerate(kfold.split(X, y)):
    print(f'LINEAR REGRESSION FOLD # {ix+1}')
    print(f'+--------------------------+')
    model = LinearRegression()
    x_train, y_train = X.iloc[train_ix, ], y.iloc[train_ix]
    x_test, y_test = X.iloc[test_ix], y.iloc[test_ix]
    print(model.fit(x_train, y_train))
    evaluate(x_train, x_test, y_train, y_test, model)

    with open(f'model/linear_regression/linear_regression_fold_{ix+1}.pkl',␣
↪'wb') as f:
        pickle.dump(model, f)
```

```
LINEAR REGRESSION FOLD # 1
+--------------------------+
LinearRegression()

Train Predictions:
    * MSE: 12035791020.18
    * RMSE: 109707.75
    * MAE: 55807.21
    * r2: 0.2875598597518624

Test Predictions:
    * MSE: 8084210857.57
    * RMSE: 89912.24
    * MAE: 52878.82
    * r2: 0.2841955888452308
-----------------------------------------------------------
```

```
LINEAR REGRESSION FOLD # 2
+--------------------------+
LinearRegression()

Train Predictions:
    * MSE: 11848696519.27
    * RMSE: 108851.72
    * MAE: 55160.55
    * r2: 0.2869747589360193

Test Predictions:
    * MSE: 9731984153.39
    * RMSE: 98650.82
    * MAE: 55604.28
    * r2: 0.29642336788376444
-----------------------------------------------------------

LINEAR REGRESSION FOLD # 3
+--------------------------+
LinearRegression()

Train Predictions:
    * MSE: 11934157990.10
    * RMSE: 109243.57
    * MAE: 55547.48
    * r2: 0.2910968482676639

Test Predictions:
    * MSE: 8973334859.72
    * RMSE: 94727.69
    * MAE: 55142.04
    * r2: 0.240662309377351
-----------------------------------------------------------

LINEAR REGRESSION FOLD # 4
+--------------------------+
LinearRegression()

Train Predictions:
    * MSE: 11972580609.95
    * RMSE: 109419.29
    * MAE: 55615.98
    * r2: 0.28947529599909205

Test Predictions:
    * MSE: 8606348392.85
    * RMSE: 92770.41
```

```
    * MAE: 54034.86
    * r2: 0.2599971582569256
------------------------------------------------------------

LINEAR REGRESSION FOLD # 5
+--------------------------+
LinearRegression()

Train Predictions:
    * MSE: 12048909988.52
    * RMSE: 109767.53
    * MAE: 56005.34
    * r2: 0.2914322854217337

Test Predictions:
    * MSE: 7898647646.47
    * RMSE: 88874.34
    * MAE: 50166.19
    * r2: 0.22557338872195354
------------------------------------------------------------

LINEAR REGRESSION FOLD # 6
+--------------------------+
LinearRegression()

Train Predictions:
    * MSE: 10658174951.07
    * RMSE: 103238.44
    * MAE: 52486.72
    * r2: 0.2876855866481024

Test Predictions:
    * MSE: 20571346118.57
    * RMSE: 143427.15
    * MAE: 62190.79
    * r2: 0.2814603415181667
------------------------------------------------------------

LINEAR REGRESSION FOLD # 7
+--------------------------+
LinearRegression()

Train Predictions:
    * MSE: 10992114032.45
    * RMSE: 104843.28
    * MAE: 53471.27
    * r2: 0.29073389975572317
```

```
Test Predictions:
    * MSE: 17499684874.56
    * RMSE: 132286.37
    * MAE: 60880.95
    * r2: 0.26794660351857746
-----------------------------------------------------------

LINEAR REGRESSION FOLD # 8
+---------------------------+
LinearRegression()

Train Predictions:
    * MSE: 11749685578.02
    * RMSE: 108395.97
    * MAE: 55645.08
    * r2: 0.2953954544263122

Test Predictions:
    * MSE: 10674310500.06
    * RMSE: 103316.55
    * MAE: 54486.45
    * r2: 0.19009122626968178
-----------------------------------------------------------

LINEAR REGRESSION FOLD # 9
+---------------------------+
LinearRegression()

Train Predictions:
    * MSE: 12196521752.03
    * RMSE: 110437.86
    * MAE: 56711.61
    * r2: 0.28886253174633636

Test Predictions:
    * MSE: 6608353551.24
    * RMSE: 81291.78
    * MAE: 42056.40
    * r2: 0.26826812318878757
-----------------------------------------------------------

LINEAR REGRESSION FOLD # 10
+---------------------------+
LinearRegression()

Train Predictions:
    * MSE: 10744735156.87
    * RMSE: 103656.81
```

```
    * MAE: 52003.49
    * r2: 0.27786337566074926

Test Predictions:
    * MSE: 20349323064.08
    * RMSE: 142651.05
    * MAE: 71214.45
    * r2: 0.27364870183300116
------------------------------------------------------------
```

[57]:

```
X
```

[57]:

```
         track_id  race_number  program_number  trakus_index   latitude  \
0               0            9              16            72  40.672901
1               0            9              16            73  40.672947
2               0            9              16            74  40.672989
3               0            9              16            63  40.672508
4               0            9              16            64  40.672554
…             …           …              …             …          …
5228425         0            9              10           167  40.672363
5228426         0            9              10           168  40.672321
5228427         0            9              10           169  40.672279
5228428         0            9              10           170  40.672241
5228429         0            9              10           171  40.672199

         longitude  distance_id  course_type  track_condition  \
0       -73.827606          600            0                2
1       -73.827591          600            0                2
2       -73.827568          600            0                2
3       -73.827782          600            0                2
4       -73.827759          600            0                2
…            …            …           …               …
5228425 -73.830856         1100            4                2
5228426 -73.830872         1100            4                2
5228427 -73.830894         1100            4                2
5228428 -73.830910         1100            4                2
5228429 -73.830933         1100            4                2

         run_up_distance  race_type  post_time  weight_carried  jockey  odds  \
0                     48          2        420             120       5  2090
1                     48          2        420             120       5  2090
2                     48          2        420             120       5  2090
3                     48          2        420             120       5  2090
4                     48          2        420             120       5  2090
…                   …        …        …              …     …    …
5228425               72          8        353             124      77  1120
```

```
5228426                72          8        353             124       77  1120
5228427                72          8        353             124       77  1120
5228428                72          8        353             124       77  1120
5228429                72          8        353             124       77  1120

         day_of_race  year  month  day
0                  5  2019      1    1
1                  5  2019      1    1
2                  5  2019      1    1
3                  5  2019      1    1
4                  5  2019      1    1
…                …    …      …   …
5228425            2  2019     11   23
5228426            2  2019     11   23
5228427            2  2019     11   23
5228428            2  2019     11   23
5228429            2  2019     11   23

[5228430 rows x 19 columns]
```

[ ]: