kaviyadevi 20106064

In [1]:
```
#to import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#to import dataset
data1=pd.read_csv(r"C:\Users\user\Downloads\15_Horse Racing Results.CSV - 15_Hors
data1
```

Out[2]:

	Dato	Track	Race Number	Distance	Surface	Prize money	Starting position	Jockey	Jockey weight	Countr
0	03.09.2017	Sha Tin	10	1400	Gress	1310000	6	K C Leung	52	Sverig
1	16.09.2017	Sha Tin	10	1400	Gress	1310000	14	C Y Ho	52	Sverig
2	14.10.2017	Sha Tin	10	1400	Gress	1310000	8	C Y Ho	52	Sverig
3	11.11.2017	Sha Tin	9	1600	Gress	1310000	13	Brett Prebble	54	Sverig
4	26.11.2017	Sha Tin	9	1600	Gress	1310000	9	C Y Ho	52	Sverig
...	...	...	...	...	...	...	...	...	...	.
27003	14.06.2020	Sha Tin	11	1200	Gress	1450000	6	A Hamelin	59	Australi
27004	21.06.2020	Sha Tin	2	1200	Gress	967000	7	K C Leung	57	Australi
27005	21.06.2020	Sha Tin	4	1200	Gress	967000	6	Blake Shinn	57	Australi
27006	21.06.2020	Sha Tin	5	1200	Gress	967000	14	Joao Moreira	57	Ne Zealan
27007	21.06.2020	Sha Tin	11	1200	Gress	1450000	7	C Schofield	55	Ne Zealan

27008 rows × 21 columns

In [3]:

```
#to display top 5 rows
data=data1.head(100)
data
```

Out[3]:

	Dato	Track	Race Number	Distance	Surface	Prize money	Starting position	Jockey	Jockey weight	Country
0	03.09.2017	Sha Tin	10	1400	Gress	1310000	6	K C Leung	52	Sverige
1	16.09.2017	Sha Tin	10	1400	Gress	1310000	14	C Y Ho	52	Sverige
2	14.10.2017	Sha Tin	10	1400	Gress	1310000	8	C Y Ho	52	Sverige
3	11.11.2017	Sha Tin	9	1600	Gress	1310000	13	Brett Prebble	54	Sverige
4	26.11.2017	Sha Tin	9	1600	Gress	1310000	9	C Y Ho	52	Sverige
...	...	...	...	...	...	...	...	...	...	...
95	10.12.2017	Sha Tin	5	1200	Gress	18500000	13	Francois-Xavier Bertras	57	Great Britain
96	10.12.2017	Sha Tin	7	1600	Gress	23000000	11	Ryan Moore	57	USA
97	01.10.2017	Sha Tin	7	1000	Gress	3000000	10	Brett Prebble	59	New Zealand
98	22.10.2017	Sha Tin	7	1200	Gress	4000000	9	Brett Prebble	59	New Zealand
99	19.11.2017	Sha Tin	7	1200	Gress	4000000	3	Brett Prebble	56	New Zealand

100 rows × 21 columns

DATA CLEANING AND PREPROCESSING

In [4]:

```
#
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 21 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Dato               100 non-null    object
 1   Track              100 non-null    object
 2   Race Number        100 non-null    int64 
 3   Distance           100 non-null    int64 
 4   Surface            100 non-null    object
 5   Prize money        100 non-null    int64 
 6   Starting position  100 non-null    int64 
 7   Jockey             100 non-null    object
 8   Jockey weight      100 non-null    int64 
 9   Country            100 non-null    object
 10  Horse age          100 non-null    int64 
 11  TrainerName        100 non-null    object
 12  Race time          100 non-null    object
 13  Path               100 non-null    int64 
 14  Final place        100 non-null    int64 
 15  FGrating           100 non-null    int64 
 16  Odds               100 non-null    object
 17  RaceType           100 non-null    object
 18  HorseId            100 non-null    int64 
 19  JockeyId           100 non-null    int64 
 20  TrainerID          100 non-null    int64 
dtypes: int64(12), object(9)
memory usage: 16.5+ KB
```

In [5]:

```
#to display summary of statistics(here to know min max value)
data.describe()
```

Out[5]:

	Race Number	Distance	Prize money	Starting position	Jockey weight	Horse age	Path	Fina
count	100.000000	100.000000	1.000000e+02	100.000000	100.000000	100.00000	100.000000	100.
mean	6.910000	1446.000000	3.562200e+06	6.170000	55.870000	6.58000	1.510000	6.
std	2.099038	334.820923	4.486259e+06	3.440857	2.942736	1.35721	1.573101	3
min	1.000000	1000.000000	9.200000e+05	1.000000	49.000000	3.00000	0.000000	1.
25%	6.000000	1200.000000	1.380000e+06	3.000000	54.000000	6.00000	0.000000	4.
50%	7.000000	1400.000000	1.950000e+06	6.000000	56.000000	7.00000	1.000000	6.
75%	8.000000	1650.000000	3.000000e+06	9.000000	58.000000	8.00000	3.000000	9.
max	10.000000	2400.000000	2.300000e+07	14.000000	60.000000	9.00000	6.000000	12.

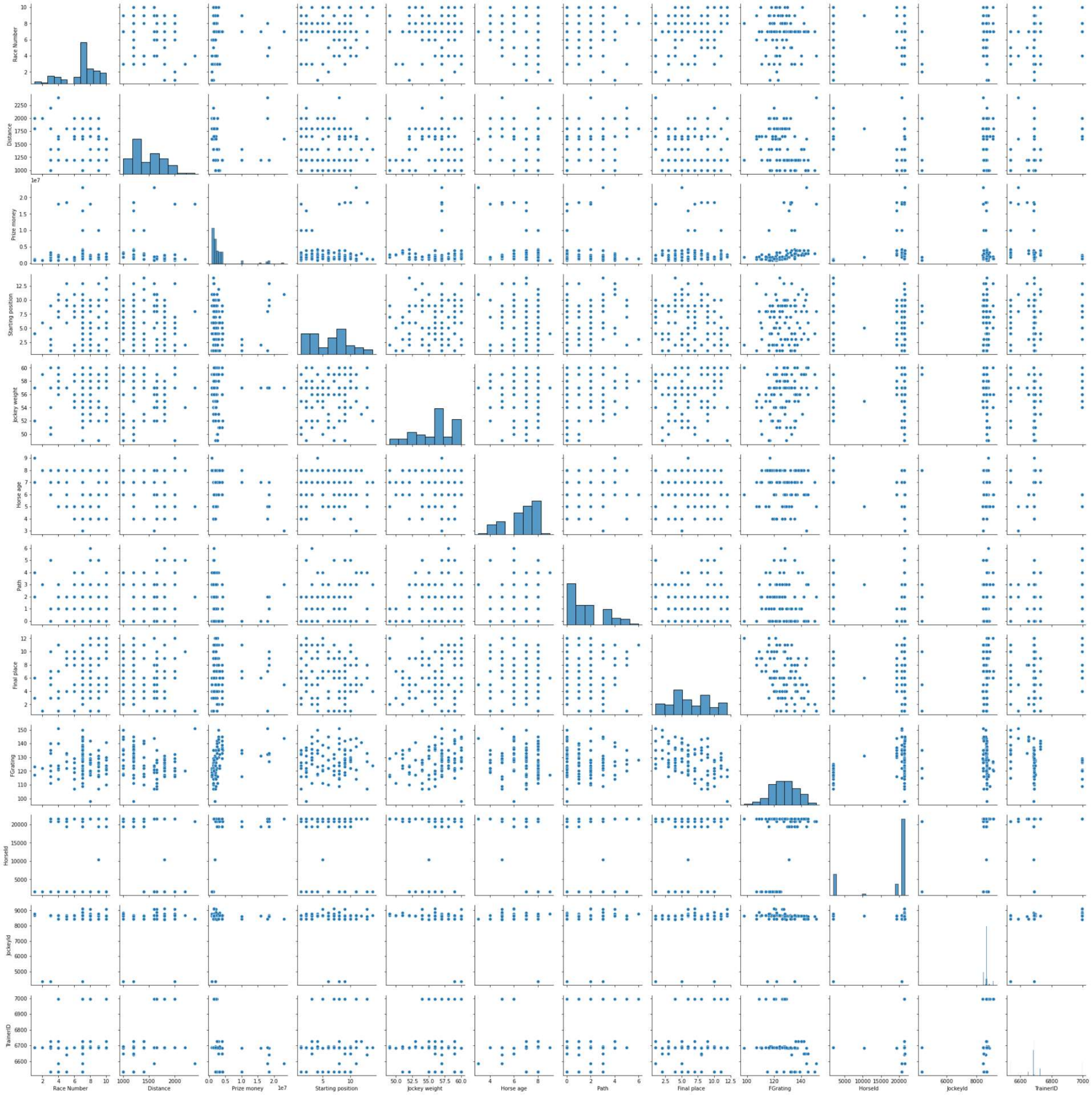
In [6]:
```
#to display the column heading
data.columns
```

Out[6]:
```
Index(['Dato', 'Track', 'Race Number', 'Distance', 'Surface', 'Prize money',
       'Starting position', 'Jockey', 'Jockey weight', 'Country', 'Horse age',
       'TrainerName', 'Race time', 'Path', 'Final place', 'FGrating', 'Odds',
       'RaceType', 'HorseId', 'JockeyId', 'TrainerID'],
      dtype='object')
```

EDA and DATA VISUALIZATION

In [7]:
```
sns.pairplot(data)
```

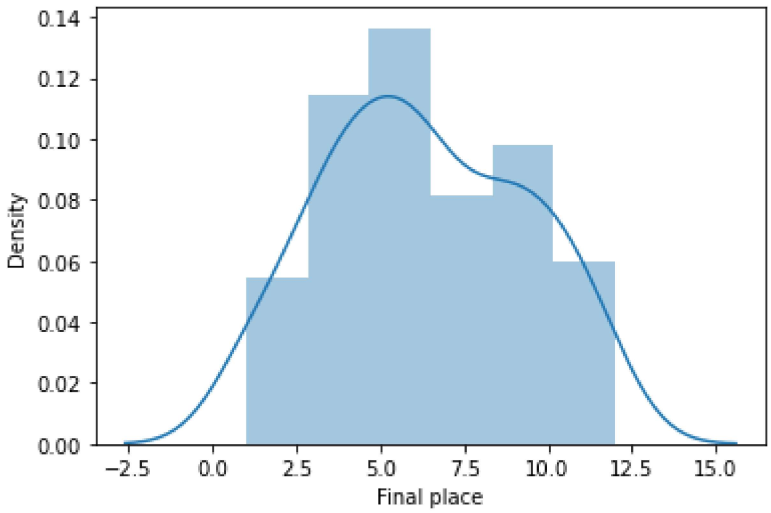
Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x21c23c75910>
```



In [8]:
```
sns.distplot(data['Final place'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Futur
eWarning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
```

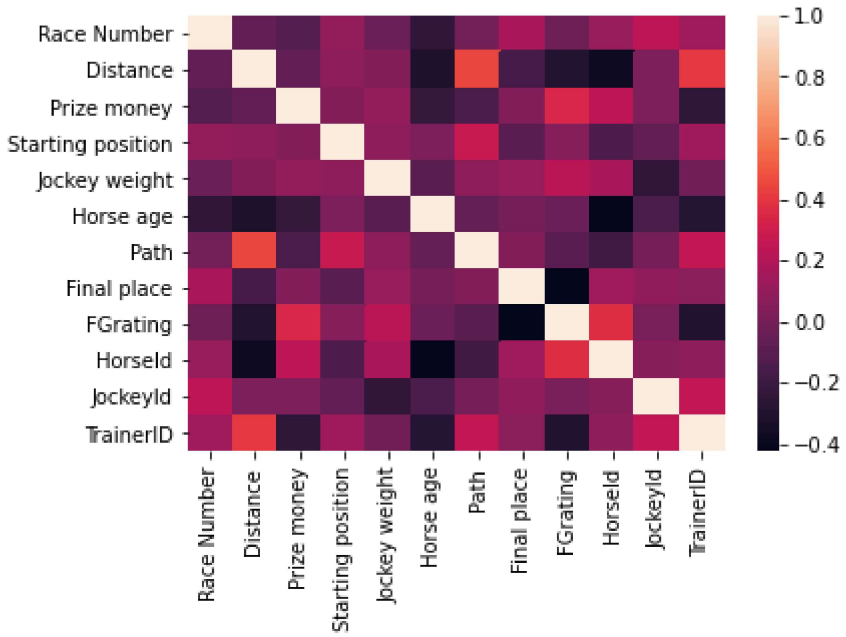
Out[8]:
```
<AxesSubplot:xlabel='Final place', ylabel='Density'>
```



In [9]:
```
df=data[['Dato', 'Track', 'Race Number', 'Distance', 'Surface', 'Prize money',
       'Starting position', 'Jockey', 'Jockey weight', 'Horse age', 'Path', 'Fina
```

In [10]:
```
sns.heatmap(df.corr())
```

Out[10]:
```
<AxesSubplot:>
```



TRAINING MODEL

In [11]:

```
x=df[['Prize money', 'Jockey weight', 'Horse age',  'Path', 'HorseId', 'JockeyId'
y=df['Final place']
```

In [12]:

```
#to split my dataset into trainning and test

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [13]:

```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[13]:

```
LinearRegression()
```

In [14]:

```
#to find intercept
print(lr.intercept_)
```

```
9.36303809762785
```

In [15]:

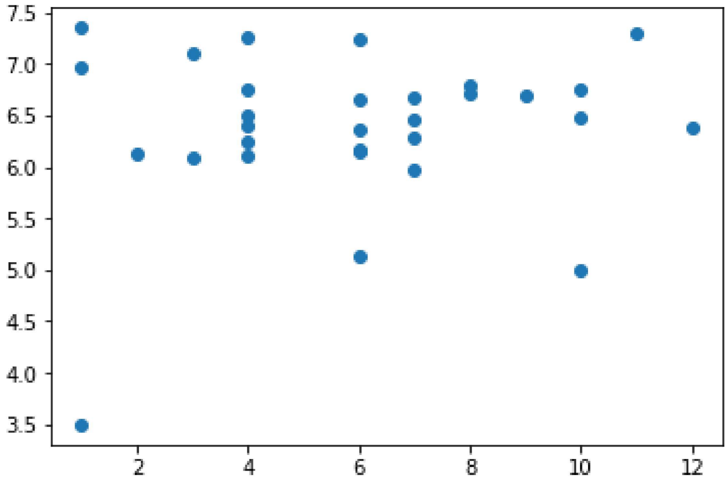
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[15]:

	Co-efficient
Prize money	7.609949e-10
Jockey weight	7.739040e-02
Horse age	1.099069e-01
Path	2.010332e-01
HorseId	7.163400e-05
JockeyId	6.222035e-04
TrainerID	-2.204513e-03

```
In [16]: prediction = lr.predict(x_test)
         plt.scatter(y_test,prediction)

Out[16]: <matplotlib.collections.PathCollection at 0x21c2d7b8700>
```



```
In [17]: print(lr.score(x_test,y_test))

         -0.03502949396008659
```

RIDGE AND LASSO REGRESSION

```
In [18]: from sklearn.linear_model import Ridge,Lasso
```

```
In [19]: rr=Ridge(alpha=10)
         rr.fit(x_train,y_train)

Out[19]: Ridge(alpha=10)
```

```
In [20]: rr.score(x_test,y_test)

Out[20]: -0.036968758402129875
```

```
In [21]: la=Lasso(alpha=10)
         la.fit(x_train,y_train)

Out[21]: Lasso(alpha=10)
```

```
In [22]: la.score(x_test,y_test)

Out[22]: -0.05353449065594229
```

```
In [24]: from sklearn.linear_model import ElasticNet
         en=ElasticNet()
         en.fit(x_train,y_train)

Out[24]: ElasticNet()
```

```
In [25]: print(en.coef_)

         [-6.39536183e-10  1.42371537e-02  0.00000000e+00  0.00000000e+00
           6.02573468e-05  5.76873207e-04 -2.10922701e-03]
```

```
In [26]: print(en.predict(x_test))

         [3.22351042 6.76993259 6.4345862  6.77236712 6.30965858 7.06892676
          5.6151163  6.229081   6.93259798 6.79868718 6.52042482 6.74670244
          6.78545053 6.63099513 6.71261943 6.90185187 6.17821072 6.73481609
          6.21962001 6.91299057 6.74506634 6.33037967 6.04416663 6.71261943
          6.76993259 4.69214    6.7788909  6.77742651 6.74173856 6.81821291]
```

```
In [27]: print(en.score(x_test,y_test))

         -0.07732682931719137
```

```
In [28]: from sklearn import metrics
```

```
In [29]: print("Mean Absolute error",metrics.mean_absolute_error(y_test,prediction))

         Mean Absolute error 2.445588877527731
```

```
In [30]: print("Mean Squared error",metrics.mean_squared_error(y_test,prediction))

         Mean Squared error 9.06340826877716
```

```
In [31]: print("Root Mean Absolute error",np.sqrt(metrics.mean_squared_error(y_test,predic

         Root Mean Absolute error 3.010549496151352
```

```
In [ ]:
```