Problem Statement

A real estate agent want help to predict the house price for regions in USA.He gave us the dataset to work on to use linear regression model.Create a model that helps him to estimate of what the house would sell for

Import libraries

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

In [2]:
```
# To import dataset
df=pd.read_csv('15 Horse csv')
df
```

Out[2]:

	Dato	Track	Race Number	Distance	Surface	Prize money	Starting position	Jockey	Jockey weight	Country	...	Tra
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	...	
...	...	...	...	...	...	...	...	...	...	...	...	
27003	14.06.2020	Sha Tin	11	1200	Gress	1450000	6	A Hamelin	59	Australia	...	
27004	21.06.2020	Sha Tin	2	1200	Gress	967000	7	K C Leung	57	Australia	...	
27005	21.06.2020	Sha Tin	4	1200	Gress	967000	6	Blake Shinn	57	Australia	...	P
27006	21.06.2020	Sha Tin	5	1200	Gress	967000	14	Joao Moreira	57	New Zealand	...	
27007	21.06.2020	Sha Tin	11	1200	Gress	1450000	7	C Schofield	55	New Zealand	...	

27008 rows × 21 columns

In [3]:
```
# To display top 10 rows
df.head(10)
```

Out[3]:

	Dato	Track	Race Number	Distance	Surface	Prize money	Starting position	Jockey	Jockey weight	Country	...	TrainerN
0	03.09.2017	Sha Tin	10	1400	Gress	1310000	6	K C Leung	52	Sverige	...	C
1	16.09.2017	Sha Tin	10	1400	Gress	1310000	14	C Y Ho	52	Sverige	...	C
2	14.10.2017	Sha Tin	10	1400	Gress	1310000	8	C Y Ho	52	Sverige	...	C
3	11.11.2017	Sha Tin	9	1600	Gress	1310000	13	Brett Prebble	54	Sverige	...	C
4	26.11.2017	Sha Tin	9	1600	Gress	1310000	9	C Y Ho	52	Sverige	...	C
5	10.12.2017	Sha Tin	1	1800	Gress	1310000	4	C Y Ho	52	Sverige	...	C
6	01.01.2018	Sha Tin	9	1800	Gress	1310000	9	C Schofield	54	Sverige	...	C
7	04.02.2018	Sha Tin	5	1800	Gress	1310000	6	Joao Moreira	57	Sverige	...	C
8	03.03.2018	Sha Tin	8	1800	Gress	1310000	3	C Y Ho	56	Sverige	...	C
9	11.03.2018	Sha Tin	10	1600	Gress	1310000	8	C Y Ho	57	Sverige	...	C

10 rows × 21 columns

Data Cleaning and Pre-Processing

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27008 entries, 0 to 27007
Data columns (total 21 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Dato               27008 non-null  object
 1   Track              27008 non-null  object
 2   Race Number        27008 non-null  int64 
 3   Distance           27008 non-null  int64 
 4   Surface            27008 non-null  object
 5   Prize money        27008 non-null  int64 
 6   Starting position  27008 non-null  int64 
 7   Jockey             27008 non-null  object
 8   Jockey weight      27008 non-null  int64 
 9   Country            27008 non-null  object
 10  Horse age          27008 non-null  int64 
 11  TrainerName        27008 non-null  object
 12  Race time          27008 non-null  object
 13  Path               27008 non-null  int64 
 14  Final place        27008 non-null  int64 
 15  FGrating           27008 non-null  int64 
 16  Odds               27008 non-null  object
 17  RaceType           27008 non-null  object
 18  HorseId            27008 non-null  int64 
 19  JockeyId           27008 non-null  int64 
 20  TrainerID          27008 non-null  int64 
dtypes: int64(12), object(9)
memory usage: 4.3+ MB
```

In [5]:
```
df.describe()
```

Out[5]:

	Race Number	Distance	Prize money	Starting position	Jockey weight	Horse age	Path
count	27008.000000	27008.000000	2.700800e+04	27008.000000	27008.000000	27008.000000	27008.000000
mean	5.268624	1401.666173	1.479445e+06	6.741447	55.867373	5.246408	1.678021
std	2.780088	276.065045	2.162109e+06	3.691071	2.737006	1.519880	1.631784
min	1.000000	1000.000000	6.600000e+05	1.000000	47.000000	2.000000	0.000000
25%	3.000000	1200.000000	9.200000e+05	4.000000	54.000000	4.000000	0.000000
50%	5.000000	1400.000000	9.670000e+05	7.000000	56.000000	5.000000	1.000000
75%	8.000000	1650.000000	1.450000e+06	10.000000	58.000000	6.000000	3.000000
max	11.000000	2400.000000	2.800000e+07	14.000000	63.000000	12.000000	11.000000

In [6]:
```
df.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')
```

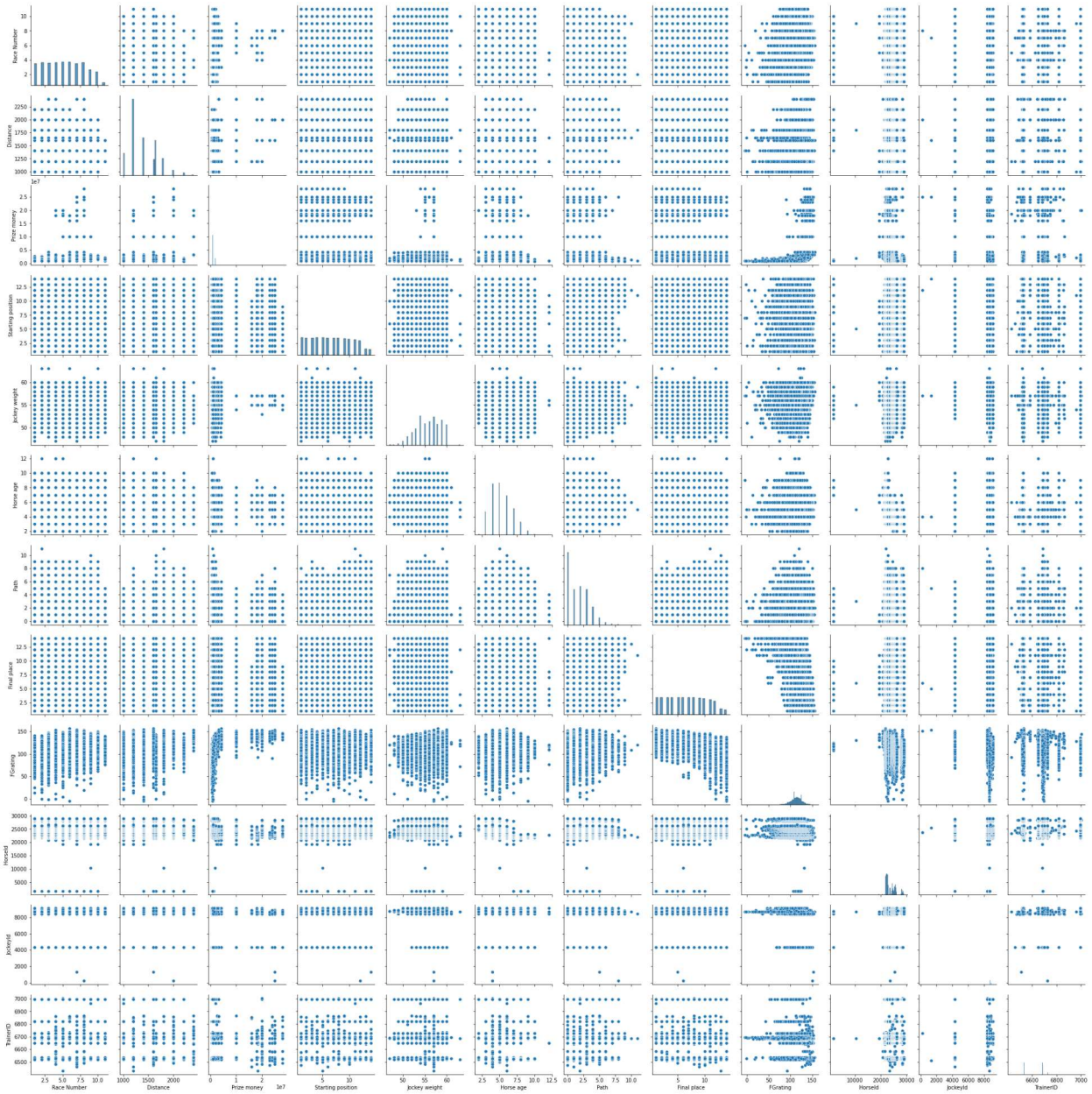
In [7]:
```
a = df.dropna(axis='columns')
a.columns
```

Out[7]:
```
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 Visualization

In [8]:
```
sns.pairplot(a)
```

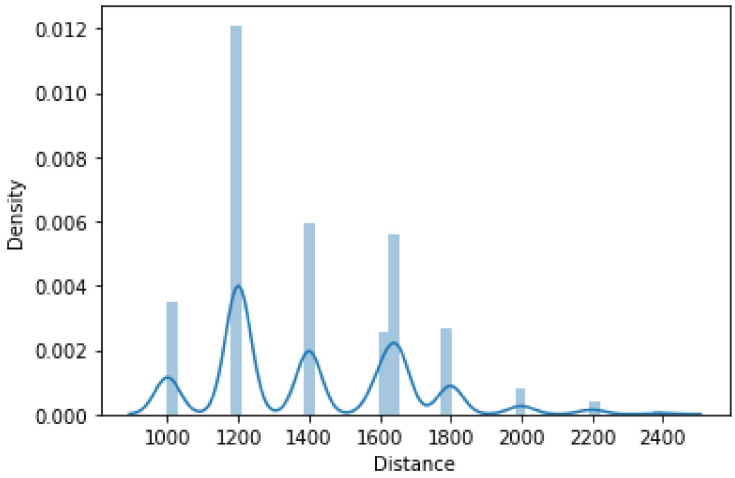
Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x2691a58b3a0>
```



In [9]:
```
sns.distplot(a['Distance'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Please
adapt your code to use either `displot` (a figure-level function with similar flexibil
ity) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

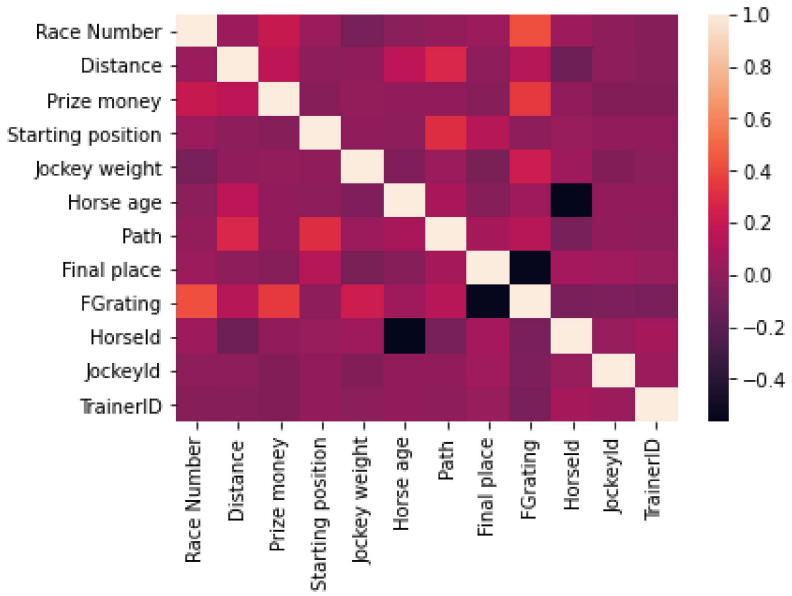
Out[9]: <AxesSubplot:xlabel='Distance', ylabel='Density'>



In [10]:
```
a1=a[[ 'Race Number', 'Distance', 'Prize money',
       'Starting position', 'Jockey weight', 'Horse age',
       'Path', 'Final place', 'FGrating',
       'HorseId', 'JockeyId', 'TrainerID']]
```

In [11]:
```
sns.heatmap(a1.corr())
```

Out[11]: <AxesSubplot:>



To Train the Model - Model Building

We are going to train Linear Regression model;We need to split out data into two variables x and y where x is independent variable (input) and y is dependent on x(output). We could ignore address column as it is

In [12]:
```
x=a1[['Race Number', 'Prize money',
       'Starting position', 'Jockey weight', 'Horse age',
       'Path', 'Final place', 'FGrating',
       'HorseId', 'JockeyId', 'TrainerID']]
y=a1['Distance']
```

To split my dataset into training and test data

In [13]:
```
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 [14]:
```
from sklearn.linear_model import LinearRegression

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

Out[14]:
```
LinearRegression()
```

In [15]:
```
print(lr.intercept_)
```

```
1927.3189790554284
```

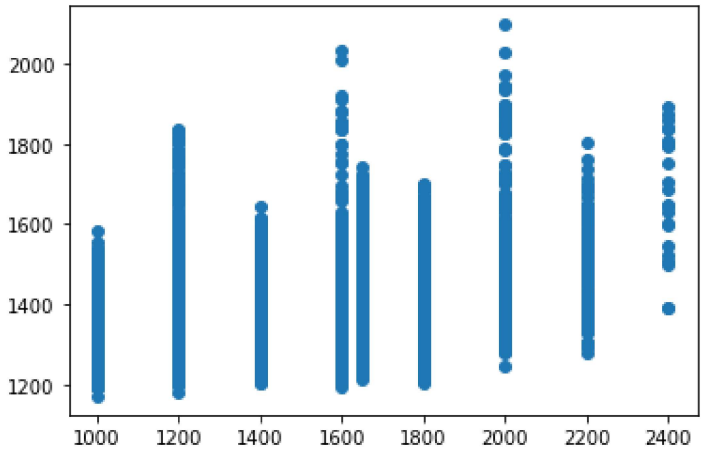
In [16]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[16]:

	Co-efficient
Race Number	-0.673872
Prize money	0.000020
Starting position	-7.059837
Jockey weight	-0.892861
Horse age	20.207805
Path	48.911251
Final place	-0.047988
FGrating	0.622915
HorseId	-0.003782
JockeyId	-0.006825
TrainerID	-0.084501

In [17]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[17]:
```
<matplotlib.collections.PathCollection at 0x269337938e0>
```



In [18]:
```
print(lr.score(x_test,y_test))
```

```
0.14108279167147864
```

ACCURACY

In [19]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [20]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
rr.score(x_test,y_test)
rr.score(x_train,y_train)
```

Out[20]:
```
0.13395852905608252
```

In [21]:
```
rr.score(x_test,y_test)
```

Out[21]:
```
0.14108046393963602
```

In [22]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[22]:
```
Lasso(alpha=10)
```

In [23]:
```
la.score(x_test,y_test)
```

Out[23]:
```
0.13824201015881998
```

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

Out[24]:
```
ElasticNet()
```

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

```
[-1.17751612e+00  1.92537395e-05 -5.62765552e+00 -9.93355548e-01
  1.54706602e+01  3.97741642e+01  7.32589951e-01  9.65894416e-01
 -6.41226254e-03 -6.30138340e-03 -7.76293329e-02]
```

In [26]:
```
print(en.intercept_)
```

```
1935.1354025941766
```

In [27]:
```
print(en.predict(x_test))
```

```
[1397.80452647 1347.59687944 1399.8361524  ... 1382.56504367 1418.93862149
 1398.68373162]
```

In [28]:
```
print(en.score(x_test,y_test))
```

```
0.1361687327257315
```

In [29]:
```
from sklearn import metrics
print("Mean Absolytre Error:",metrics.mean_absolute_error(y_test,prediction))
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction))
```

```
Mean Absolytre Error: 214.86384817114887
Mean Squared Error: 66809.69010795571
Root Mean Squared Error: 258.47570506327224
```