Prediction of monthly Henry Hub natural gas spot prices using 4 machine learning algorithms; Support Vector Machines, Random Forest Regression, Gradient Boosting Machine and Artificial Neural Networks

The Henry Hub natural gas price is an important benchmark in the natural gas industry because it is based on the supply and demand of natural gas as an independent commodity unlike other hub prices that create a pricing system considering natural gas as a product of oil and thus indexing its price to oil.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

The data was gathered independently from the EIA website for the period between January 2001 and November 2021. The features considered were:

- Cooling Degree Days
- Heating Degree Days
- Natural Gas Demand
- Natural Gas Imports
- Natural Gas Exports
- Natural Gas Drilling Rigs Count
- Natural Gas Supply
- Natural Gas Storage
- West Texas Intermediate (WTI) oil price
- Heating oil price
- USD/EUR exchange rate

```
df=pd.read_csv('natural_gas_data.csv')
df.head()
```

	month	cool_days	hot_days	demand	imports	exports	rig_count	supply	stoı
0	2001-01	4.0	928.0	2676998.0	373077.0	25547.0	879.0	1753237.0	56094

EXPLORATORY DATA ANALYSIS AND DATA CLEANING

The dataset contains 251 samples alongside 11 features.

```
df.shape
```

```
(251, 13)
```

```
df.describe()
```

	cool_days	hot_days	demand	imports	exports	rig_count	
count	251.000000	251.000000	2.510000e+02	251.000000	251.000000	251.000000	2
mean	116.840637	351.892430	2.114223e+06	288789.003984	163093.075697	667.804781	2
std	123.940173	310.865425	4.608949e+05	58537.633529	140154.507369	470.567366	4
min	3.000000	3.000000	1.368369e+06	174225.000000	23637.000000	70.000000	1
25%	15.000000	39.500000	1.742105e+06	238303.500000	63901.500000	190.500000	1
50%	52.000000	284.000000	2.067048e+06	282159.000000	117329.000000	704.000000	1
75%	220.500000	629.000000	2.400512e+06	334006.500000	198450.500000	989.500000	2
max	404.000000	969.000000	3.424302e+06	426534.000000	595411.000000	1585.000000	3

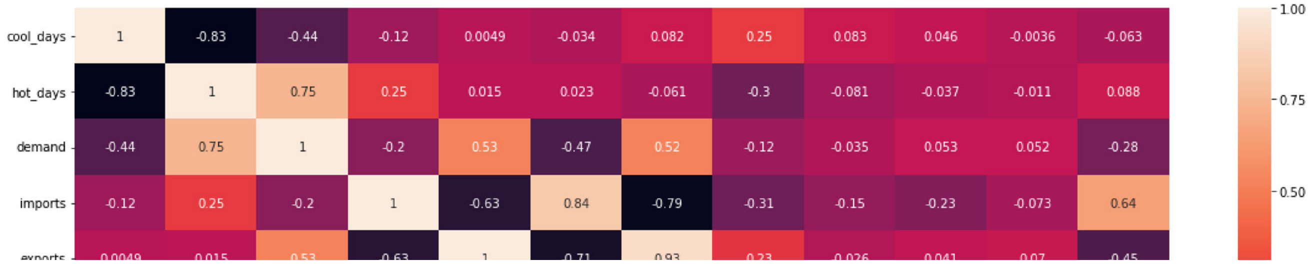
```
df.skew()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Dropping
  """Entry point for launching an IPython kernel.
cool_days      0.882533
hot_days       0.402952
demand         0.631137
imports        0.161730
exports        1.502598
rig_count      0.298495
supply         0.747028
storage       -0.229896
wti_price      0.363692
heating_oil    0.318235
usd_rate       0.981133
gas_price      1.481865
dtype: float64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 251 entries, 0 to 250
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   month        251 non-null    object 
 1   cool_days    251 non-null    float64
 2   hot_days     251 non-null    float64
 3   demand       251 non-null    float64
 4   imports      251 non-null    float64
 5   exports      251 non-null    float64
 6   rig_count    251 non-null    float64
 7   supply       251 non-null    float64
 8   storage      251 non-null    float64
 9   wti_price    251 non-null    float64
 10  heating_oil  251 non-null    float64
 11  usd_rate     251 non-null    float64
 12  gas_price    251 non-null    float64
dtypes: float64(12), object(1)
memory usage: 25.6+ KB
```

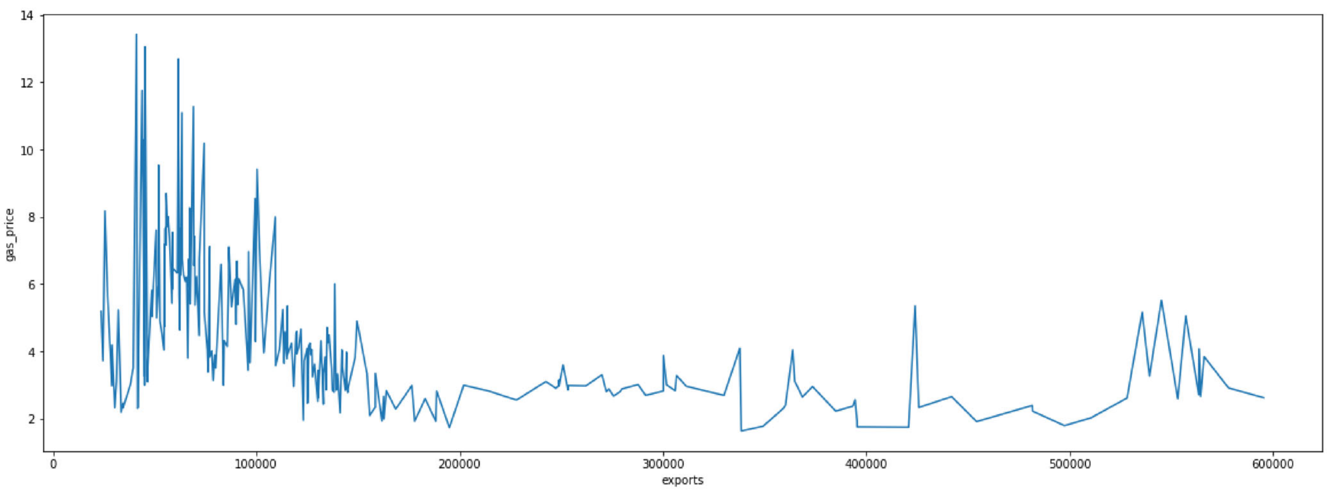
```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), annot=True)
plt.show()
```



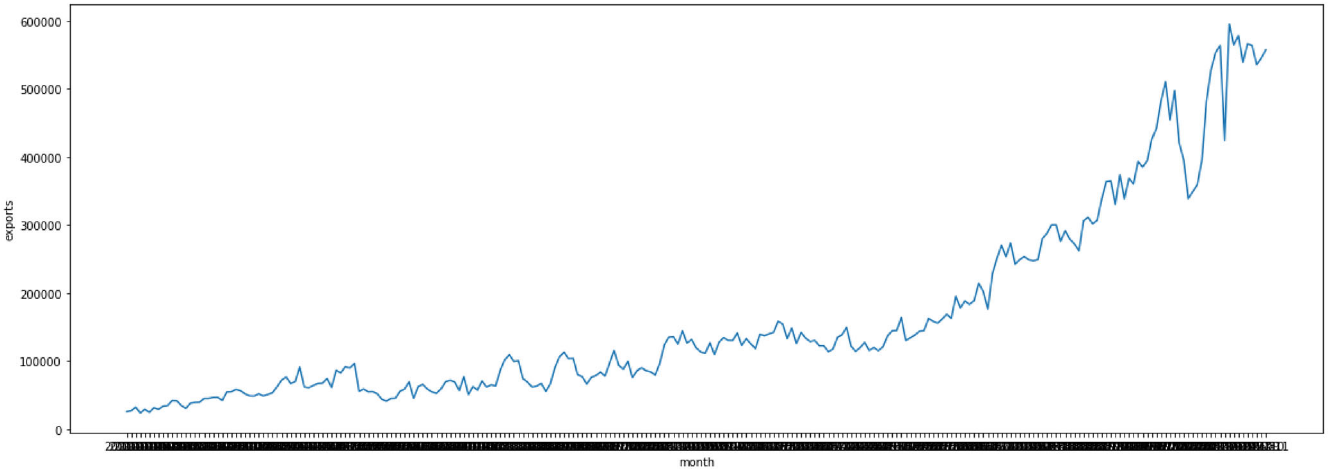
Write a function that automatically creates line graphs of two selected features in the dataframe

```
def graph(a,b):
    plt.figure(figsize=(20,7))
    sns.lineplot(x=a, y=b)
    plt.show()
```

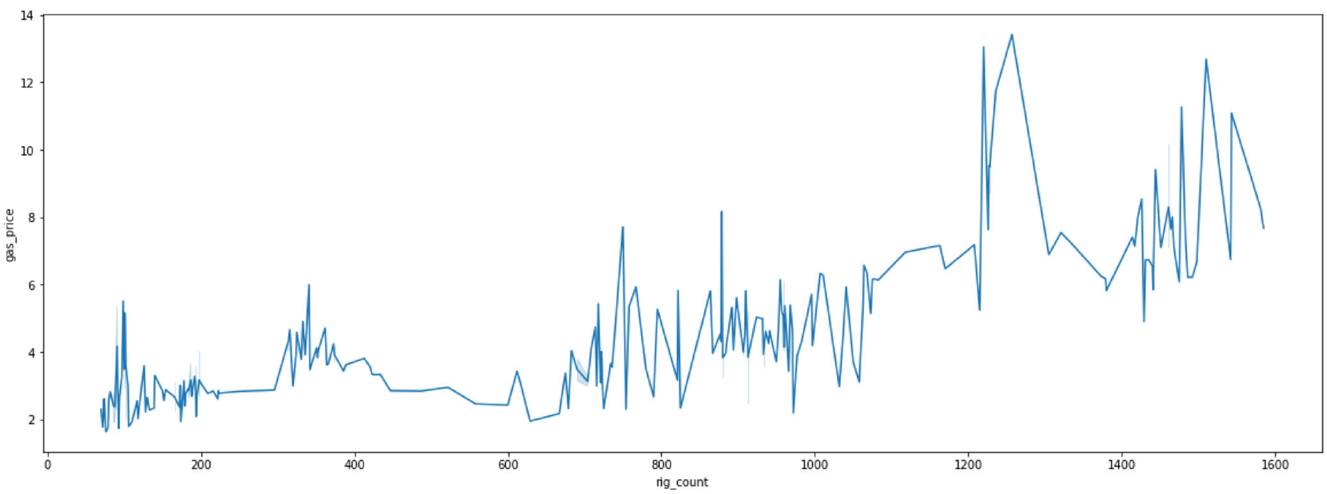
```
graph(df['exports'], df['gas_price'])
```




```
graph(df['month'], df['exports'])
```




```
graph(df['rig_count'], df['gas_price'])
```



MODEL BUILDING

Write a function that returns predictions for each value which can be used for each model.

```
def get_preds(y_test, y_preds):
    y_test=pd.DataFrame(y_test)
    y_test.rename(columns={0:'Actual'}, inplace=True)
```

```
    y_preds=pd.DataFrame(y_preds)
    y_preds.rename(columns={0:'Predicted'}, inplace=True)
    predictions=pd.concat([y_test, y_preds], axis=1)
    return predictions
```

```
X=df.iloc[:, 1:-1].values
y=df.iloc[:, -1].values
```

```
X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.2, random_state=42)
```

Support Vector Regression

Feature scaling is necessary for optimal performance of the SVR algorithm. Standardization is thus implemented on the dataset as a feature scaling technique.

```
from sklearn.model_selection import train_test_split, cross_val_score,KFold, GridSearchCV
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
```

Make a copy of the preexisting dataframe by using the .copy() method

```
df1=df.copy()
```

```
df1.head()
```

	month	cool_days	hot_days	demand	imports	exports	rig_count	supply	sto
0	2001-01	4.0	928.0	2676998.0	373077.0	25547.0	879.0	1753237.0	56094
1	2001-02	14.0	720.0	2309464.0	328289.0	26882.0	898.0	1582557.0	52408
2	2001-03	13.0	663.0	2246633.0	358103.0	32121.0	913.0	1766754.0	50419

```
X1=df1.iloc[:, 1:-1].values
y1=df1.iloc[:, -1].values.reshape(-1,1)
X_train1, X_test1, y_train1, y_test1= train_test_split(X1, y1, test_size=0.2, random_state=42
sc=StandardScaler()
X_train1=sc.fit_transform(X_train1)
```

```
X_test1=sc.transform(X_test1)
sc_y=StandardScaler()
y_train1=sc_y.fit_transform(y_train1)
y_test1=sc_y.transform(y_test1)
```

Hyperparameter tuning is an important step in model building in order to fully maximize the model's prediction abilities. We will be making use of the Grid Search Cross Validation technique for this cause.

```
reg_sv=SVR()
p_grid={'C':[ 1000, 10000,100000], 'kernel':['rbf', 'poly']}
search=GridSearchCV(estimator=reg_sv, param_grid=p_grid)
search.fit(X_train1, y_train1.ravel())
sv_preds=search.best_estimator_
y_preds=sv_preds.predict(X_test1)
print('The RMSE score for the SVR model is', np.sqrt(mean_squared_error(y_test1, y_preds)))
```

```
The RMSE score for the SVR model is 0.3824840968344997
```

Use a 10-fold cross validation technique for model validation. A decision to run the process 30 times was taken to further study the RMSE value in as many random cases as possible.

```
for i in range(30):
  outer_cv=KFold(n_splits=10, shuffle= True)
  scores = cross_val_score(sv_preds, X_train1, y_train1.ravel(), scoring='neg_root_mean_squar
  print(np.mean(scores))
```

```
-0.38243451859616345
-0.40509101868398495
-0.350317206831854
-0.3973550041003397
-0.3603995727315042
-0.35041612278746687
-0.3948785829860342
-0.36552622726166206
-0.3707524988209293
-0.392455772703256
-0.40983888551523345
-0.3628906331238545
-0.3600044040239342
-0.39097786294814785
-0.3643754560133593
```

```
-0.3713692550304593
-0.36336217232890444
-0.3702443043599001
-0.3841980184371805
-0.3862681546297513
-0.3643816397326459
-0.3730239132380471
-0.373653839765291
-0.38682338344929806
-0.370144115908464
-0.37968993186334465
-0.37281036646422105
-0.4332349492567696
-0.3778660388469087
-0.38604736279663665
```

```
y_test2 = sc_y.inverse_transform(y_test1)
pre1 = sc_y.inverse_transform(y_preds.reshape(-1,1))
svr_predictions=get_preds(y_test2, pre1)
svr_predictions
```

	Actual	Predicted
0	4.24	3.462038
1	3.11	2.866313
2	3.92	3.807978
3	8.69	7.909211
4	4.80	5.066917
5	1.73	1.764678
6	2.98	2.964551
7	5.16	4.231195
8	2.46	2.342300
9	3.71	3.957812
10	2.77	2.608679
11	2.69	2.788427
12	2.65	2.622163
13	2.84	2.782357
14	7.14	5.846306
15	2.33	2.730958
16	3.10	2.523928
17	3.43	2.931379
18	4.49	6.513004
19	5.43	6.616361
20	5.35	3.174262
21	3.09	3.454795
22	4.32	3.517138
23	4.90	6.258579
24	2.34	2.327985
25	6.35	6.011734
26	9.53	7.269502
27	2.34	2.989617
28	5.03	5.976293
29	1.77	2.314853

30	2.39	2.108457
31	4.63	3.798999
32	2.64	2.444596
33	3.15	2.407132

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
```

```
rf=RandomForestRegressor()
rf.fit(X_train, y_train)
y_pred=rf.predict(X_test)
print('The RMSE score for the RFR model is', np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
The RMSE score for the RFR model is 0.8583886910380183
```

41	4.52	4.838861

```
for i in range(30):
  outer_cv=KFold(n_splits=10, shuffle= True)
  scores = cross_val_score(rf, X, y, scoring='neg_root_mean_squared_error', cv=outer_cv)
  print(np.mean(scores))
```

```
-0.8706765547211622
-0.8520472069328321
-0.8674147491022837
-0.8692423368767856
-0.8330875444260826
-0.8461092754173556
-0.8110950146564452
-0.9593094128252915
-0.9100032913947468
-0.8711628534547102
-0.8724053316736505
-0.8343878008425311
-0.8585130269130217
-0.8194350154823752
-0.8087943487635879
-0.8730593913285004
-0.8642503126802552
-0.8627290569444884
-0.8248277680976672
-0.9117927418022853
-0.8516242887020864
-0.842326834048715
-0.8389748031550452
-0.8652397042620074
-0.8807110070714697
-0.8608975502417413
-0.8373927103827212
-0.897291795818951
-0.8736661725129847
-0.8951126242888728
```

```
rf_predictions=get_preds(y_test, y_pred)
rf_predictions
```

	Actual	Predicted
0	4.24	6.1369
1	3.11	6.7833
2	3.92	6.4975
3	8.69	2.6311
4	4.80	7.5289
5	1.73	3.3030
6	2.98	2.8114
7	5.16	4.9009
8	2.46	11.1814
9	3.71	2.8035
10	2.77	3.8324
11	2.69	3.8229
12	2.65	2.6852
13	2.84	2.4813
14	7.14	3.5948
15	2.33	9.6981
16	3.10	7.1185
17	3.43	1.9536
18	4.49	2.0587
19	5.43	2.8718
20	5.35	6.7836
21	3.09	5.5056
22	4.32	3.9279

Gradient Boosting Machine

```
from sklearn.ensemble import GradientBoostingRegressor
```

```
gb=GradientBoostingRegressor()
gb.fit(X_train, y_train)
gb_pred=gb.predict(X_test)
print('The RMSE score for the GBR model is', np.sqrt(mean_squared_error(y_test, gb_pred)))
```

```
The RMSE score for the GBR model is 0.8540470649589342
31    4.03    4.2043
```

```
for i in range(20):
  outer_cv=KFold(n_splits=10, shuffle= True)
  scores = cross_val_score(gb, X, y, scoring='neg_root_mean_squared_error', cv=outer_cv)
  print(np.mean(scores))
```

```
-0.8437703952282559
-0.84121231272826
-0.7859000830346204
-0.7790010773929379
-0.7755438628686597
-0.8160477474671801
-0.777575532233348
-0.8697214061308195
-0.8242851839855525
-0.8620721123751138
-0.8181425789314071
-0.8155905390959252
-0.872190197963287
-0.9178881380014416
-0.7816743099520338
-0.8179416108368794
-0.7227081861900496
-0.8864756329737908
-0.8068022136024947
-0.7902468829318188
```

```
gbr_predictions=get_preds(y_test, gb_pred)
gbr_predictions
```

	Actual	Predicted
0	4.24	4.008750
1	3.11	3.938106
2	3.92	3.753718
3	8.69	7.842604
4	4.80	3.982835
5	1.73	2.118093
6	2.98	2.950029
7	5.16	3.553646
8	2.46	3.096789
9	3.71	3.837725
10	2.77	2.308644
11	2.69	2.862736
12	2.65	2.972596
13	2.84	3.010094
14	7.14	6.763701
15	2.33	2.834572
16	3.10	2.793708
17	3.43	3.263724
18	4.49	5.193133
19	5.43	4.511549
20	5.35	2.881012
21	3.09	2.954511
22	4.32	3.698285
23	4.90	6.873124
24	2.34	2.595345
25	6.35	5.671593
26	9.53	7.554886
27	2.34	3.174620
28	5.03	5.663948
29	1.77	2.359047

30	2.39	2.182622
31	4.63	5.024157
32	2.64	2.811626
33	3.15	2.600814
34	7.71	4.797014
35	4.42	3.803490
36	2.99	3.350100
37	8.00	7.435553
38	2.67	3.534922
39	7.10	7.618053

Artificial Neural Network

```
import tensorflow as tf
from keras.wrappers.scikit_learn import KerasRegressor
```

```
tf.__version__
```

```
'2.8.0'
```

```
ann=tf.keras.models.Sequential()
```

We use a 3 hidden layer neural network with 256 units alongside the rectified linear activation function

```
ann.add(tf.keras.layers.Dense(units=256, activation='relu'))
ann.add(tf.keras.layers.Dense(units=256, activation='relu'))
ann.add(tf.keras.layers.Dense(units=256, activation='relu'))
ann.add(tf.keras.layers.Dense(units=256, activation='relu'))
```

```
ann.add(tf.keras.layers.Dense(units=1, activation='linear'))
```

```
ann.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_squared_error'])
```

```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=10)
```

Model validation for the Artificial neural network is necessary to check for overfitting which the ANN is known to be very susceptible to.

```
history=ann.fit(X_train1, y_train1, batch_size=32, epochs=300, validation_data=(X_train1, y_t
```

```
7/7 [==============================] - 0s 14ms/step - loss: 0.0025 - mean_squared_err
Epoch 11/300
7/7 [==============================] - 0s 14ms/step - loss: 0.0021 - mean_squared_err
Epoch 12/300
7/7 [==============================] - 0s 16ms/step - loss: 0.0023 - mean_squared_err
Epoch 13/300
7/7 [==============================] - 0s 15ms/step - loss: 0.0021 - mean_squared_err
Epoch 14/300
7/7 [==============================] - 0s 13ms/step - loss: 0.0016 - mean_squared_err
Epoch 15/300
7/7 [==============================] - 0s 13ms/step - loss: 0.0013 - mean_squared_err
Epoch 16/300
7/7 [==============================] - 0s 16ms/step - loss: 8.1409e-04 - mean_squarec
Epoch 17/300
7/7 [==============================] - 0s 14ms/step - loss: 6.6181e-04 - mean_squarec
Epoch 18/300
7/7 [==============================] - 0s 15ms/step - loss: 6.1123e-04 - mean_squarec
Epoch 19/300
7/7 [==============================] - 0s 13ms/step - loss: 6.1425e-04 - mean_squarec
Epoch 20/300
7/7 [==============================] - 0s 12ms/step - loss: 3.7451e-04 - mean_squarec
Epoch 21/300
7/7 [==============================] - 0s 13ms/step - loss: 4.1306e-04 - mean_squarec
Epoch 22/300
7/7 [==============================] - 0s 16ms/step - loss: 3.5033e-04 - mean_squarec
Epoch 23/300
7/7 [==============================] - 0s 13ms/step - loss: 3.6137e-04 - mean_squarec
Epoch 24/300
7/7 [==============================] - 0s 16ms/step - loss: 3.7025e-04 - mean_squarec
Epoch 25/300
7/7 [==============================] - 0s 13ms/step - loss: 4.7019e-04 - mean_squarec
Epoch 26/300
7/7 [==============================] - 0s 17ms/step - loss: 3.1941e-04 - mean_squarec
Epoch 27/300
7/7 [==============================] - 0s 18ms/step - loss: 3.2089e-04 - mean_squarec
Epoch 28/300
7/7 [==============================] - 0s 13ms/step - loss: 2.7641e-04 - mean_squarec
Epoch 29/300
7/7 [==============================] - 0s 12ms/step - loss: 2.6426e-04 - mean_squarec
Epoch 30/300
7/7 [==============================] - 0s 13ms/step - loss: 2.4841e-04 - mean_squarec
Epoch 31/300
7/7 [==============================] - 0s 12ms/step - loss: 2.0251e-04 - mean_squarec
Epoch 32/300
7/7 [==============================] - 0s 15ms/step - loss: 2.8302e-04 - mean_squarec
Epoch 33/300
7/7 [==============================] - 0s 13ms/step - loss: 2.8717e-04 - mean_squarec
Epoch 34/300
7/7 [==============================] - 0s 11ms/step - loss: 3.1679e-04 - mean_squarec
Epoch 35/300
```

```
Epoch 35/300
7/7 [==============================] - 0s 11ms/step - loss: 4.0220e-04 - mean_squarec
Epoch 36/300
7/7 [==============================] - 0s 13ms/step - loss: 4.7149e-04 - mean_squarec
Epoch 37/300
7/7 [==============================] - 0s 13ms/step - loss: 5.5069e-04 - mean_squarec
Epoch 38/300
7/7 [==============================] - 0s 13ms/step - loss: 6.6893e-04 - mean_squarec
```

```
ann_preds=ann.predict(X_test1)
```

```
mse = tf.keras.losses.MeanSquaredError()
ann_mse=mse(y_test1, ann_preds).numpy()
ann_mse
```

```
0.0790542
```

```
print('The RMSE score for the ANN model is', np.sqrt(ann_mse))
```

```
The RMSE score for the ANN model is 0.28116578
```

A plot showing the drop in the loss between the training and validation sets through the epochs

```
plt.figure(figsize=(20,6))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='best')
plt.show()
```