In [8]:

```
import math
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import hashlib
```

In [9]:

```
# Importing Classifier Modules
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.neural_network import MLPClassifier

from sklearn.model_selection import learning_curve
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import preprocessing
from sklearn.model_selection import validation_curve
from sklearn.model_selection import learning_curve
```

In [10]:
```
def plot_learning_curve(
    estimator,
    title,
    X,
    y,
    axes=None,
    ylim=None,
    cv=None,
    n_jobs=None,
    train_sizes=np.linspace(0.1, 1.0, 5),
):
    """
    Generate 3 plots: the test and training learning curve, the trai
    samples vs fit times curve, the fit times vs score curve.

    Parameters
    ----------
    estimator : estimator instance
        An estimator instance implementing `fit` and `predict` metho
        will be cloned for each validation.

    title : str
        Title for the chart.

    X : array-like of shape (n_samples, n_features)
        Training vector, where ``n_samples`` is the number of sample
        ``n_features`` is the number of features.

    y : array-like of shape (n_samples) or (n_samples, n_features)
        Target relative to ``X`` for classification or regression;
        None for unsupervised learning.

    axes : array-like of shape (3,), default=None
        Axes to use for plotting the curves.

    ylim : tuple of shape (2,), default=None
        Defines minimum and maximum y-values plotted, e.g. (ymin, ym

    cv : int, cross-validation generator or an iterable, default=Non
        Determines the cross-validation splitting strategy.
        Possible inputs for cv are:

          - None, to use the default 5-fold cross-validation,
          - integer, to specify the number of folds.
          - :term:`CV splitter`,
          - An iterable yielding (train, test) splits as arrays of i

        For integer/None inputs, if ``y`` is binary or multiclass,
        :class:`StratifiedKFold` used. If the estimator is not a cla
        or if ``y`` is neither binary nor multiclass, :class:`KFold`

        Refer :ref:`User Guide <cross_validation>` for the various
        cross-validators that can be used here.

    n_jobs : int or None, default=None
        Number of jobs to run in parallel.
```

```
        ``None`` means 1 unless in a :obj:`joblib.parallel_backend`
        ``-1`` means using all processors. See :term:`Glossary <n_jo
        for more details.

    train_sizes : array-like of shape (n_ticks,)
        Relative or absolute numbers of training examples that will
        generate the learning curve. If the ``dtype`` is float, it i
        as a fraction of the maximum size of the training set (that
        determined by the selected validation method), i.e. it has t
        (0, 1]. Otherwise it is interpreted as absolute sizes of the
        sets. Note that for classification the number of samples usu
        to be big enough to contain at least one sample from each cl
        (default: np.linspace(0.1, 1.0, 5))
    """
    if axes is None:
        _, axes = plt.subplots(1, 3, figsize=(20, 5))

    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = learning_
        estimator,
        X,
        y,
        cv=cv,
        n_jobs=n_jobs,
        train_sizes=train_sizes,
        return_times=True,
    )
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes[0].grid()
    axes[0].fill_between(
        train_sizes,
        train_scores_mean - train_scores_std,
        train_scores_mean + train_scores_std,
        alpha=0.1,
        color="r",
    )
    axes[0].fill_between(
        train_sizes,
        test_scores_mean - test_scores_std,
        test_scores_mean + test_scores_std,
        alpha=0.1,
        color="g",
    )
    axes[0].plot(
        train_sizes, train_scores_mean, "o-", color="r", label="Trai
    )
```

```
    axes[0].plot(
        train_sizes, test_scores_mean, "o-", color="g", label="Cross
    )
    axes[0].legend(loc="best")

    # Plot n_samples vs fit_times
    axes[1].grid()
    axes[1].plot(train_sizes, fit_times_mean, "o-")
    axes[1].fill_between(
        train_sizes,
        fit_times_mean - fit_times_std,
        fit_times_mean + fit_times_std,
        alpha=0.1,
    )
    axes[1].set_xlabel("Training examples")
    axes[1].set_ylabel("fit_times")
    axes[1].set_title("Scalability of the model")

    # Plot fit_time vs score
    # Plot n_samples vs fit_times
    # Plot learning curve
    fit_time_argsort = fit_times_mean.argsort()
    fit_time_sorted = fit_times_mean[fit_time_argsort]
    test_scores_mean_sorted = test_scores_mean[fit_time_argsort]
    test_scores_std_sorted = test_scores_std[fit_time_argsort]
    axes[2].grid()
    axes[2].plot(fit_time_sorted, test_scores_mean_sorted, "o-")
    axes[2].fill_between(
        fit_time_sorted,
        test_scores_mean_sorted - test_scores_std_sorted,
        test_scores_mean_sorted + test_scores_std_sorted,
        alpha=0.1,
    )
    axes[2].set_xlabel("fit_times")
    axes[2].set_ylabel("Score")
    axes[2].set_title("Performance of the model")

    return plt

```

In [11]:
```
pokemon = pd.read_csv("pokemon.csv") # Dataset pokemon stats
combats = pd.read_csv("combats.csv") # Datas:qXet pokemon battles and

pokemon["Type 2"] = pokemon["Type 2"].fillna("NA")
pokemon.head()

```

Out[11]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False
3	4	Mega Venusaur	Grass	Poison	80	100	123	122	120	80	1	False
4	5	Charmander	Fire	NA	39	52	43	60	50	65	1	False

In [12]:
```
pokemon.columns
```

Out[12]:
```
Index(['#', 'Name', 'Type 1', 'Type 2', 'HP', 'Attack', 'Defense', 'Sp.
Atk',
       'Sp. Def', 'Speed', 'Generation', 'Legendary'],
      dtype='object')
```

Data preprocessing

First approach processing

We wanted to make some comparisons according to our beliefs of what is more impactfull in the outcome of a pokemon battle and for that we need a base comparison, we guided this first approach to an approach seen in [here (https://github.com/kartikeya-rana/pokemon_battle/blob/master/Pokemon.ipynb)](https://github.com/kartikeya-rana/pokemon_battle/blob/master/Pokemon.ipynb), in this approach the individual opted to create a new data set that consisted of pokemon 1 stats followed by pokemon 2 statistics and finnaly the winner of the battle, 1 in case the first pokemon wins 0 otherwise, for simplicity purposes it is not possible to have a match with no winner.

Second approach processing

After that we thought about instead of concatenating both pokemons statistics we wanted to make a subtraction of the first pokemon with the second and compare the new results with the first ones, we also eliminated both the generation and the Legendary information, as well as the type information, since that we know that pokemons of different generation have the same overall raw power, and we also know that pokemon that are legendary do not have big boosts in their stats, and they do not win against every non legendary pokemon

Third approach processing

This last approach had in account our knowledge of the domain, we believe that pokemons with higher speed and higher attack are more prone to win, therefore we extracted those values and included them in a new dataset.

In [13]:

```
data_one_hot_encoding = []
extra_data_one_hot_encoding = []
i = 0


# for each tuple of combats.csv
for t in combats.itertuples():
    i += 1
    first_pokemon = t[1] # get the first pokemon
    second_pokemon = t[2] # get the second pokemon
    winner = t[3]         # get the winner

    if i <= 5001:
        x = pokemon.loc[pokemon["#"]==first_pokemon].values[:, 2:][0]
        y = pokemon.loc[pokemon["#"]==second_pokemon].values[:, 2:][0
        #diff = (x-y)[:6] # difference between "base stats hp...."
        z = np.concatenate((x,y))
        if winner == first_pokemon:
            z = np.append(z, [0])
        else:
            z = np.append(z, [1])


        data_one_hot_encoding.append(z)
    elif i < 10000:
        x = pokemon.loc[pokemon["#"]==first_pokemon].values[:, 2:][0]
        y = pokemon.loc[pokemon["#"]==second_pokemon].values[:, 2:][0
        #diff = (x-y)[:6] # difference between "base stats hp...."
        z = np.concatenate((x,y))
        if winner == first_pokemon:
            z = np.append(z, [0])
        else:
            z = np.append(z, [1])


        extra_data_one_hot_encoding.append(z)
    else:
        break



data_one_hot_encoding = np.asarray(data_one_hot_encoding)
```

In [14]:

```
data_diff_base_stats = []
extra_data_diff_base_stats = []
i = 0


# for each tuple of combats.csv
for t in combats.itertuples():
    i += 1
    first_pokemon = t[1] # get the first pokemon
    second_pokemon = t[2] # get the second pokemon
    winner = t[3]         # get the winner

    if i <= 5001:
        x = pokemon.loc[pokemon["#"]==first_pokemon].values[:, 2:][0]
        y = pokemon.loc[pokemon["#"]==second_pokemon].values[:, 2:][0
        diff = (x[2:8]-y[2:8]) # difference between "base stats hp...

        z = []
        z = np.append(diff,z,0)


        if winner == first_pokemon:
            z = np.append(z, [0])
        else:
            z = np.append(z, [1])

        data_diff_base_stats.append(z)

    elif i < 10000:
        x = pokemon.loc[pokemon["#"]==first_pokemon].values[:, 2:][0]
        y = pokemon.loc[pokemon["#"]==second_pokemon].values[:, 2:][0
        diff = (x[2:8]-y[2:8]) # difference between "base stats hp...

        z = []
        z = np.append(diff,z,0)


        if winner == first_pokemon:
            z = np.append(z, [0])
        else:
            z = np.append(z, [1])

        extra_data_diff_base_stats.append(z)
    else:
        break

data_diff_base_stats = np.asarray(data_diff_base_stats)
data_diff_base_stats = data_diff_base_stats[:, :-1].astype(int)

extra_data_diff_base_stats = np.asarray(extra_data_diff_base_stats)
```

In [15]:
```
pokemon_with_types = pokemon.copy()
types_list = [x for x in pokemon["Type 2"].unique()] + [x for x in po

types_list = list(set(types_list))

print(types_list.remove("NA"))
# make NA -> 0
types_list = ["NA"] + types_list

types_map = { x : types_list.index(x) for x in types_list}

pokemon_with_types["Type 1"] = pokemon_with_types["Type 1"].map(types
pokemon_with_types["Type 2"] = pokemon_with_types["Type 2"].map(types

pokemon.head()

print(types_map)
print(types_list)
```

```
None
{'NA': 0, 'Water': 1, 'Psychic': 2, 'Poison': 3, 'Bug': 4, 'Ghost': 5,
'Electric': 6, 'Fire': 7, 'Steel': 8, 'Flying': 9, 'Rock': 10, 'Normal':
11, 'Fighting': 12, 'Ground': 13, 'Dragon': 14, 'Fairy': 15, 'Dark': 16,
'Grass': 17, 'Ice': 18}
['NA', 'Water', 'Psychic', 'Poison', 'Bug', 'Ghost', 'Electric', 'Fire',
'Steel', 'Flying', 'Rock', 'Normal', 'Fighting', 'Ground', 'Dragon', 'Fa
iry', 'Dark', 'Grass', 'Ice']
```

In [16]:

```
data_with_types = []
extra_data_with_types = []
i = 0


# for each tuple of combats.csv
for t in combats.itertuples():
    i += 1
    first_pokemon = t[1] # get the first pokemon
    second_pokemon = t[2] # get the second pokemon
    winner = t[3]         # get the winner

    if i <= 5001:
        x = pokemon_with_types.loc[pokemon["#"]==first_pokemon]
        x = x.drop(columns=["Name","#"]).values[0]
        y = pokemon_with_types.loc[pokemon["#"]==second_pokemon].drop


        z = np.concatenate((x,y))

        if winner == first_pokemon:
            z = np.append(z, [0])
        else:
            z = np.append(z, [1])

        data_with_types.append(z)
    elif i < 10000:
        x = pokemon_with_types.loc[pokemon["#"]==first_pokemon]
        x = x.drop(columns=["Name","#"]).values[0]
        y = pokemon_with_types.loc[pokemon["#"]==second_pokemon].drop


        z = np.concatenate((x,y))

        if winner == first_pokemon:
            z = np.append(z, [0])
        else:
            z = np.append(z, [1])

        extra_data_with_types.append(z)
    else:
        break


data_with_types = np.asarray(data_with_types)
data_with_types= pd.DataFrame(data_with_types, columns=[
       'Type 1', 'Type 2', 'HP', 'Attack', 'Defense', 'Sp. Atk',
       'Sp. Def', 'Speed', 'Generation', 'Legendary' ,
       'Type 1', 'Type 2', 'HP', 'Attack', 'Defense', 'Sp. Atk',
       'Sp. Def', 'Speed', 'Generation', 'Legendary','Win'] )

data_with_types = data_with_types.astype({
       'Type 1':'int32', 'Type 2':'int32', 'HP':'int32', 'Attack':'ir
       'Sp. Def':'int32', 'Speed':'int32', 'Generation':'int32', 'Leg
       'Type 1':'int32', 'Type 2':'int32', 'HP':'int32', 'Attack':'ir
       'Sp. Def':'int32', 'Speed':'int32', 'Generation':'int32', 'Leg
```

```

extra_data_with_types = np.asarray(extra_data_with_types)
extra_data_with_types= pd.DataFrame(extra_data_with_types, columns=[
        'Type 1', 'Type 2', 'HP', 'Attack', 'Defense', 'Sp. Atk',
        'Sp. Def', 'Speed', 'Generation', 'Legendary' ,
        'Type 1', 'Type 2', 'HP', 'Attack', 'Defense', 'Sp. Atk',
        'Sp. Def', 'Speed', 'Generation', 'Legendary','Win'] )

extra_data_with_types = extra_data_with_types.astype({
        'Type 1':'int32', 'Type 2':'int32', 'HP':'int32', 'Attack':'ir
        'Sp. Def':'int32', 'Speed':'int32', 'Generation':'int32', 'Leg
        'Type 1':'int32', 'Type 2':'int32', 'HP':'int32', 'Attack':'ir
        'Sp. Def':'int32', 'Speed':'int32', 'Generation':'int32', 'Leg



X_imp_feat = data_with_types[['Type 1', 'Type 2','Attack', 'Speed']]

y = data_with_types["Win"]

print(y)



```

```
0       1
1       1
2       1
3       1
4       0
       ..
4996    0
4997    1
4998    1
4999    0
5000    1
Name: Win, Length: 5001, dtype: int32
```

In [17]:

```
extra_data_with_types.head()
```

Out[17]:

	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	...	Type 2	HP	At
0	8	0	40	55	70	45	60	30	5	0	...	0	38	
1	5	17	85	110	76	65	82	56	6	0	...	0	30	
2	18	9	90	85	100	95	125	85	1	1	...	1	110	
3	11	0	70	70	70	70	70	70	3	0	...	0	95	
4	11	9	85	120	70	50	60	100	4	0	...	3	40	

5 rows × 21 columns

Data correlation number of wins

number of wins is correlated with speed and attack being special attack and defense are also important factors as well as being legendary Contrary to my beliefs generations are quite well balanced, since there is almos no correlation between the 2 We can also see that the base stats do not have much correlation with generation, therefore pokemons that belong to diferent generations are on an equal foot in terms of base power.

Data correlation legendary or not

Once again through this correlation heatmap it is possible to say that generation as no important role to classify a pokemon as legendary, one thing that strikes out is Sp. Attack and Attack as well as Sp. Defense this seem to be somehow correlated to classifing a pokemon as legendary or not

In [18]:

```
unique_ids=pokemon['#']
wins_by_id = []
for _id in unique_ids:
    wins_by_id.append([_id ,(combats["Winner"] == _id).sum() ])

wins_by_id = np.asarray(wins_by_id)

pokemon["Wins"] = wins_by_id[:,1]
pokemon["Legendary"] = pokemon["Legendary"].astype(int)



```

In [19]:
```python

plt.figure(figsize=(20,8))
sns.heatmap(pokemon.corr(),square=True, vmin=-1, vmax=1, annot=True)
```

Out[19]: <AxesSubplot:>

Data correlation number of wins

number of wins is correlated with speed and attack being special attack and defense are also important factors as well as being legendary Contrary to my beliefs generations are quite well balanced, since there is almos no correlation between the 2 We can also see that the base stats do not have much correlation with generation, therefore pokemons that belong to diferent generations are on an equal foot in terms of base power.

Data correlation legendary or not

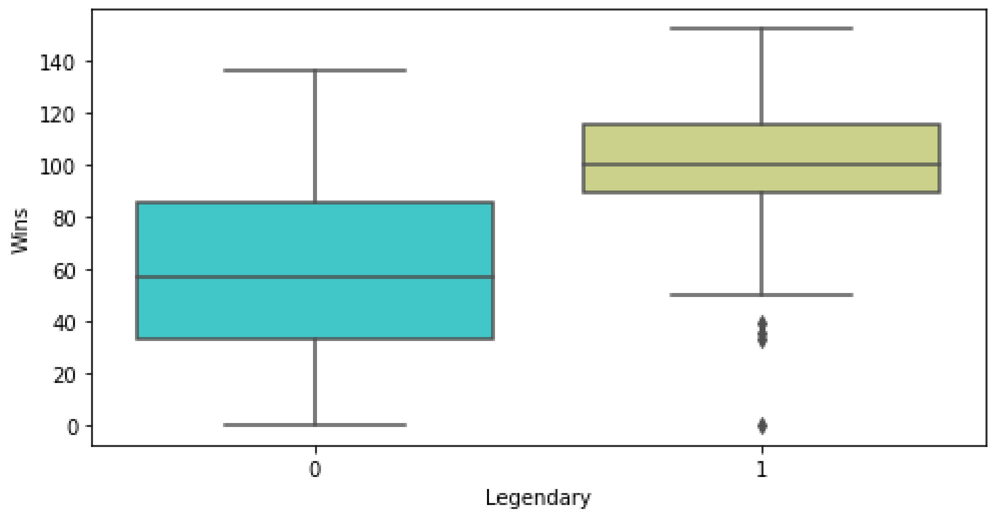
Once again through this correlation heatmap it is possible to say that generation as no important role to classify a pokemon as legendary, one thing that strikes out is Sp. Attack and Attack as well as Sp. Defense this seem to be somehow correlated to classifing a pokemon as legendary or not

Graph that shows the correlation between number of wins and speed

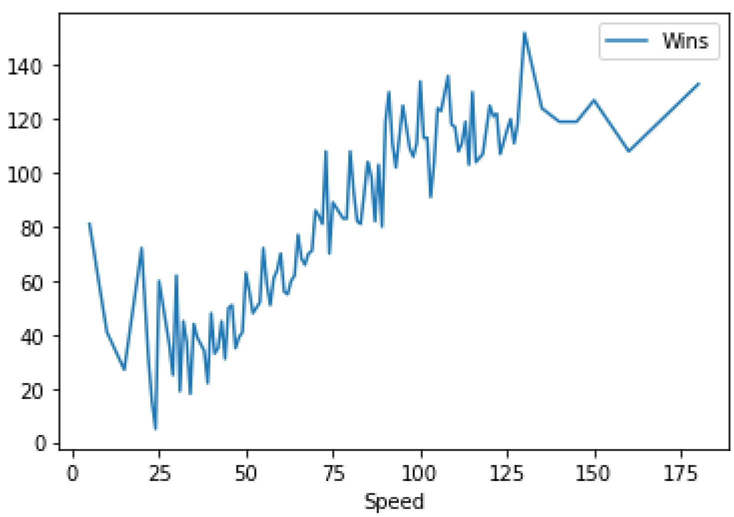
Overall the speed of the pokemon is more impactfull in the outcome of battle than most of the other attributes.

In [20]:
```
#boxplot of Attack vs. Legendary
plt.figure(figsize=(8, 4))
sns.boxplot(x='Legendary',y='Wins',data=pokemon, palette='rainbow')

#stripplot of speed Wins relation
plt.figure(figsize=(20,8))
speed_wins = pokemon[['Speed','Wins']].groupby(['Speed'], as_index=Fa
speed_wins.sort_values(by='Speed',ascending=True).plot(kind='line')
#stripplot of Attack vs. Legendary, palette='rainbow'
plt.figure(figsize=(20,8))
speed_wins = pokemon[['Attack','Wins']].groupby(['Attack'], as_index=
speed_wins.sort_values(by='Attack',ascending=True).plot(kind='line')

```

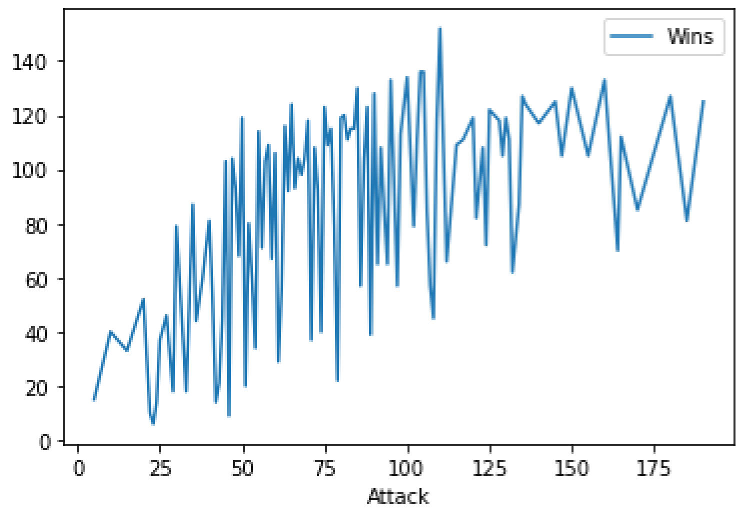
Out[20]: <AxesSubplot:xlabel='Attack'>



<Figure size 1440x576 with 0 Axes>



<Figure size 1440x576 with 0 Axes>

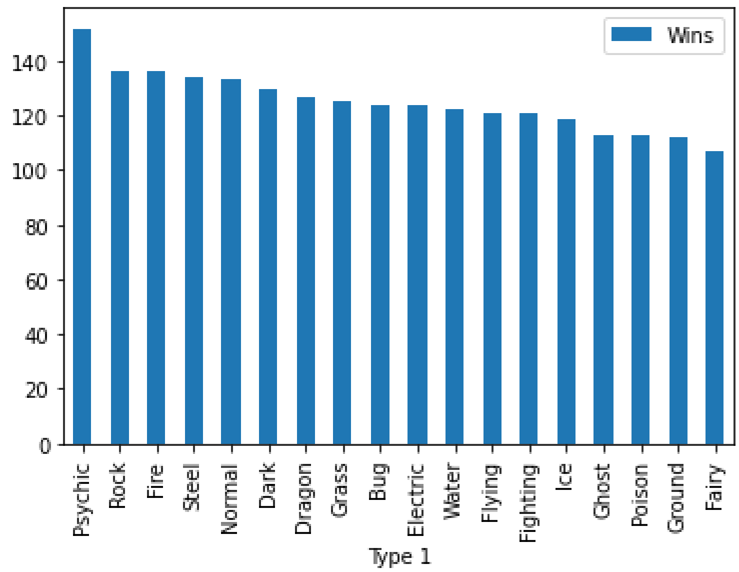


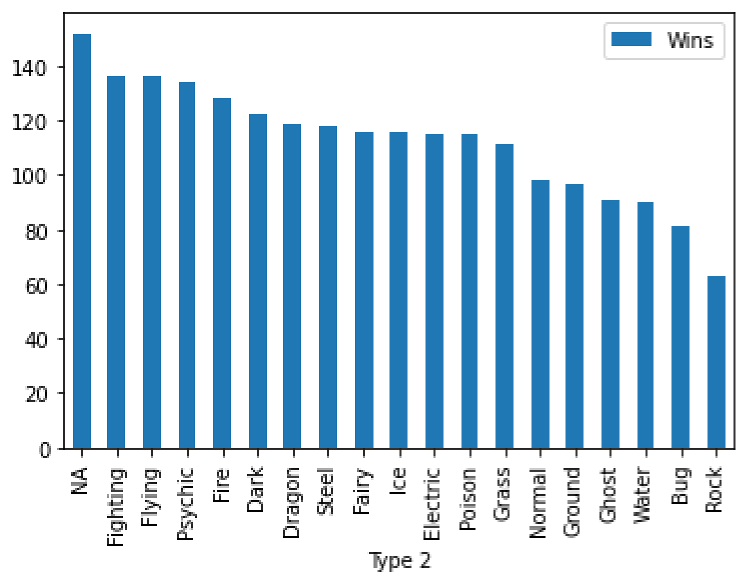
1 type pokemon go brr cause strong

In [21]:

```
type_1 = pokemon[['Type 1','Wins']].groupby(['Type 1'], as_index=Fals
type_1.sort_values(by='Wins',ascending=False).plot(kind='bar')

type_2 = pokemon[['Type 2','Wins']].groupby(['Type 2'], as_index=Fals
type_2.sort_values(by='Wins',ascending=False).plot(kind='bar')
```

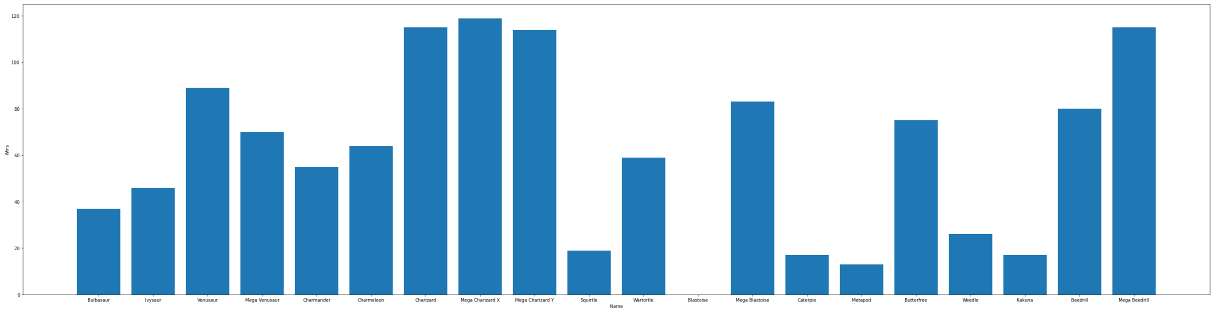
Out[21]: <AxesSubplot:xlabel='Type 2'>





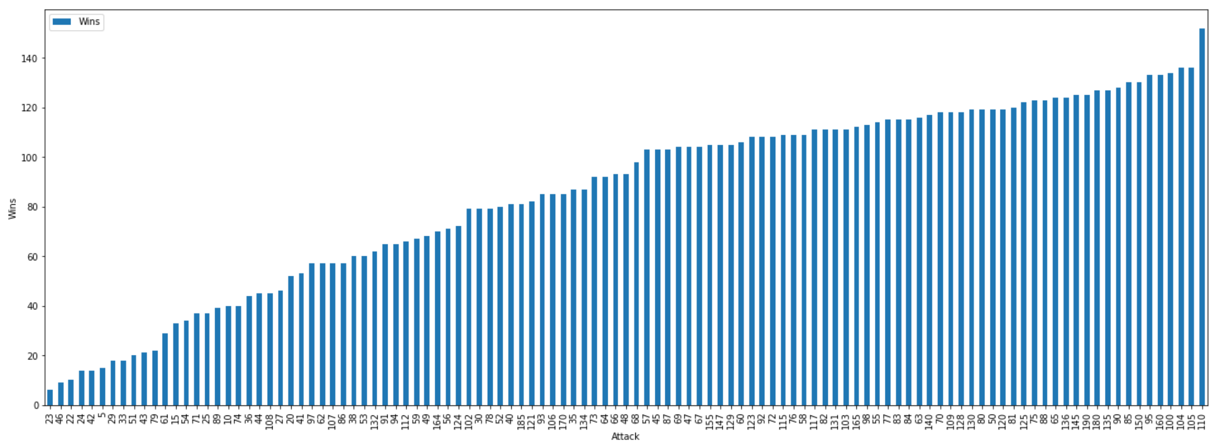
In [22]:
```
plt.figure(figsize=(48, 12))
plt.xlabel("Name")
plt.ylabel("Wins")
plt.bar(pokemon['Name'][:20], pokemon['Wins'][:20])
```

Out[22]: <BarContainer object of 20 artists>



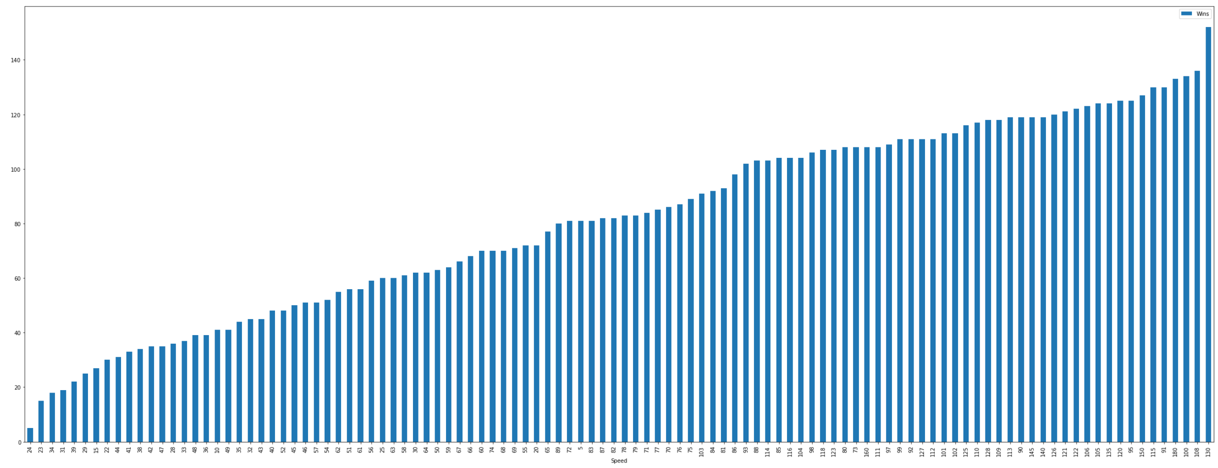
In [23]:
```

type_1 = pokemon[['Attack','Wins']].groupby(['Attack'], as_index=Fals
type_1.sort_values(by='Wins',ascending=True).plot(kind='bar', figsize
#Number of wins is directly proportional too attack damage
```

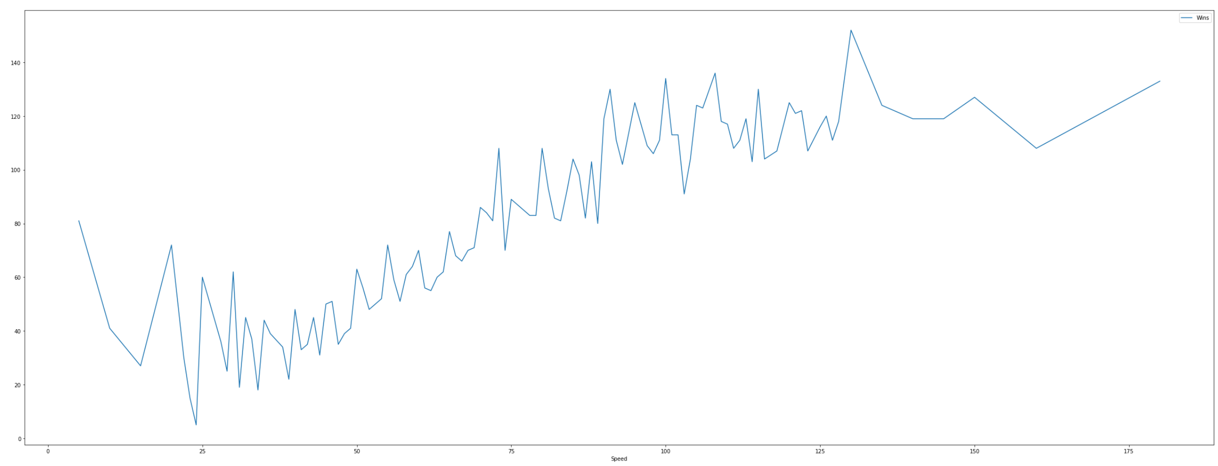
Out[23]: Text(0, 0.5, 'Wins')



In [24]:
```
type_1 = pokemon[['Speed','Wins']].groupby(['Speed'], as_index=False)
type_1.sort_values(by='Wins',ascending=True).plot(kind='bar', figsize
#Number of wins is directly proportional too Speed
type_1 = pokemon[['Speed','Wins']].groupby(['Speed'], as_index=False)
type_1.sort_values(by='Speed',ascending=True).plot(kind='line', figsi
#Number of wins is directly proportional too Speed
```

Out[24]: <AxesSubplot:xlabel='Speed'>





Classifier Results Not Having in Account Types

As we know types take an important role in pokemon battles, for example a electric pokemon deals 2 times more damage to an water pokemon than to a fire pokemon and to times less damage to a rock pokemon, this interactions might prove challenging to some classification algorithms.

Second approach

Features usados HP Diff int64 Attack Diff int64 Defens Diff int64 Sp. Atk diff int64 Sp. Def dif int64 Speed diff int64

In [25]:
```
data_pd_frame = pd.DataFrame(data_diff_base_stats, columns=["HP Diff"
# "HP Diff & Attack Diff & Defens Diff & Sp. Atk diff & Sp. Def dif &
# & -20 & -6 & 10 & -15 & 10 & -19
for column in data_pd_frame.columns:
    data_pd_frame[column] = data_pd_frame[column].astype(int)


extra_data_pd_frame = pd.DataFrame(extra_data_diff_base_stats, columr
# "HP Diff & Attack Diff & Defens Diff & Sp. Atk diff & Sp. Def dif &
# & -20 & -6 & 10 & -15 & 10 & -19
for column in data_pd_frame.columns:
    extra_data_pd_frame[column] = extra_data_pd_frame[column].astype(
```

In [26]:
```
data_pd_frame.dtypes
data_pd_frame.head(1)
```

Out[26]:

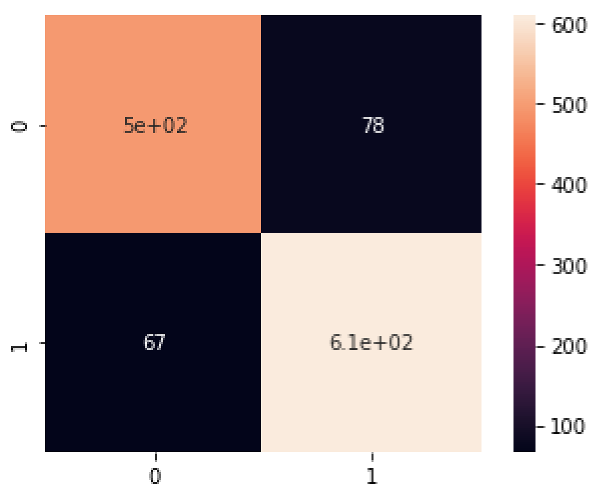
	HP Diff	Attack Diff	Defens Diff	Sp. Atk diff	Sp. Def dif	Speed diff
0	-20	-6	10	-15	10	-19

In [27]:
```
X_diff_stats = data_pd_frame.values

X_diff_train, X_diff_test, y_train, y_test = train_test_split(X_diff_
```

In [28]:
```
# Logistic Regression
clf = LogisticRegression(max_iter=10000)
clf.fit(X_diff_train, y_train)
y_pred_log_reg = clf.predict(X_diff_test)
acc_log_reg = round( clf.score(X_diff_test, y_test) * 100, 2)
print(str(acc_log_reg) + ' percent')
sns.heatmap(confusion_matrix(y_test, y_pred_log_reg), square=True, ar
```

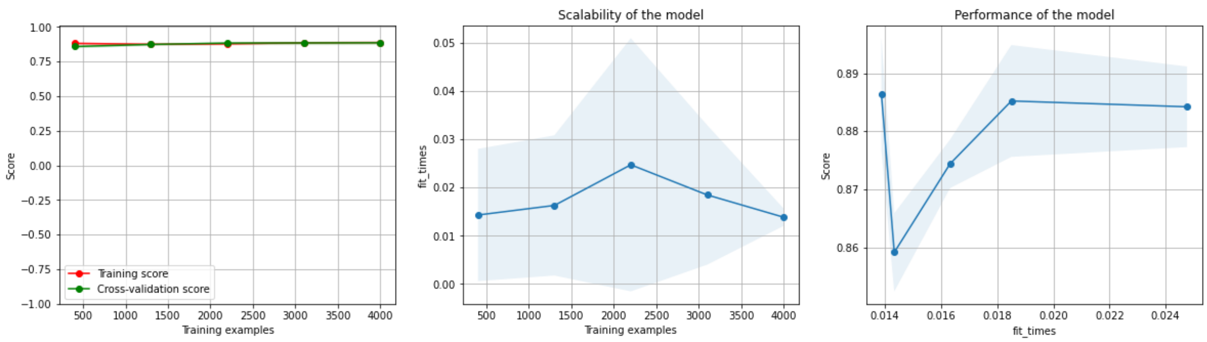
```
88.41 percent
```

Out[28]: <AxesSubplot:>



In [29]:
```
plot_learning_curve(
    clf, "Logistic Regression", X_diff_stats, y,ylim=(-1, 1.01), cv=5
)
```

Out[29]:
```
<module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packages/matplotlib/pyplot.py'>
```



In [30]:
```
pd.DataFrame(zip(data_pd_frame.columns, np.transpose(clf.coef_[0])),
```

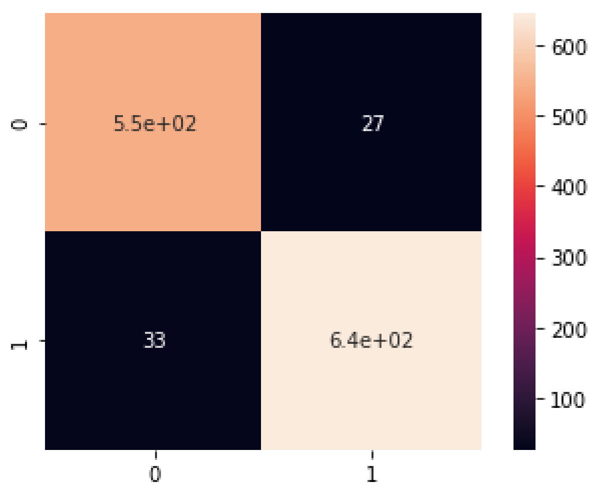
Out[30]:

	features	coef
0	HP Diff	-0.000941
1	Attack Diff	-0.009897
2	Defens Diff	-0.003292
3	Sp. Atk diff	0.002448
4	Sp. Def dif	-0.001284
5	Speed diff	-0.061518

In [31]:
```
# Random Forest Classifier difference between most imp features
# Every diff
clf = RandomForestClassifier(n_estimators=100)
model = clf.fit(X_diff_train,y_train)
pred = model.predict(X_diff_test)
# print('Accuracy of {}:'.format(name), accuracy_score(pred, y_test))
print('Accuracy :', accuracy_score(pred, y_test))
print(classification_report(y_test, pred))
sns.heatmap(confusion_matrix(y_test, pred), square=True, annot=True)
```

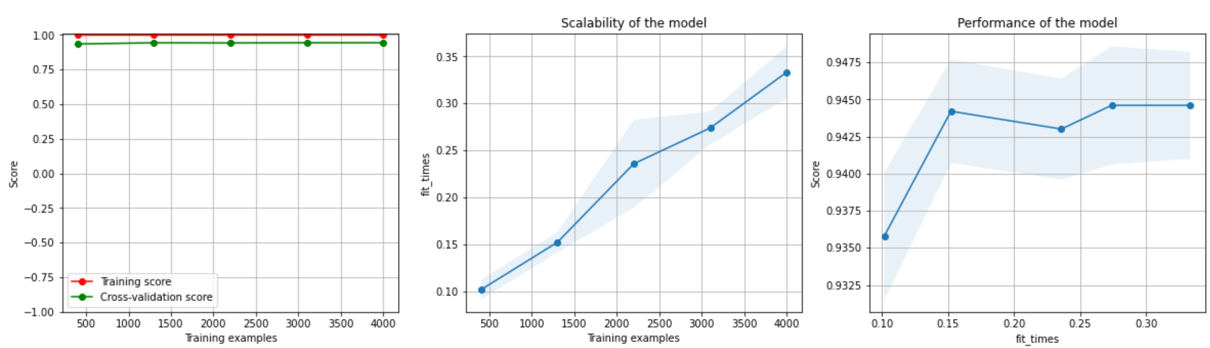
```
Accuracy : 0.9520383693045563
              precision    recall  f1-score   support

           0       0.94      0.95      0.95       574
           1       0.96      0.95      0.96       677

    accuracy                           0.95      1251
   macro avg       0.95      0.95      0.95      1251
weighted avg       0.95      0.95      0.95      1251
```

Out[31]: <AxesSubplot:>



In [32]:
```
plot_learning_curve(
    clf, "Random Forest Classifier", X_diff_stats, y,ylim=(-1, 1.01),
)
```

Out[32]: <module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packages/matplotlib/pyplot.py'>

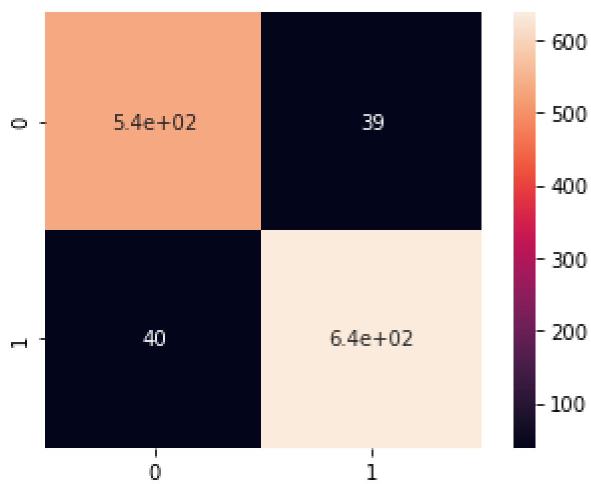


Leave Only Attack and Speed diff

In [33]:
```
X_Most_Imp_Features = data_pd_frame.drop(columns=["HP Diff", "Sp. Atk
X_train, X_test, y_train, y_test = train_test_split(X_Most_Imp_Featur
randomFlorestclf = RandomForestClassifier(n_estimators=100)
model = randomFlorestclf.fit(X_train,y_train)
pred = model.predict(X_test)
# print('Accuracy of {}:'.format(name), accuracy_score(pred, y_test))
print('Accuracy :', accuracy_score(pred, y_test))
print(classification_report(y_test, pred))
sns.heatmap(confusion_matrix(y_test,pred), square=True, annot=True)
```

```
Accuracy : 0.9368505195843325
              precision    recall  f1-score   support

           0       0.93      0.93      0.93       574
           1       0.94      0.94      0.94       677

    accuracy                           0.94      1251
   macro avg       0.94      0.94      0.94      1251
weighted avg       0.94      0.94      0.94      1251
```

Out[33]: <AxesSubplot:>



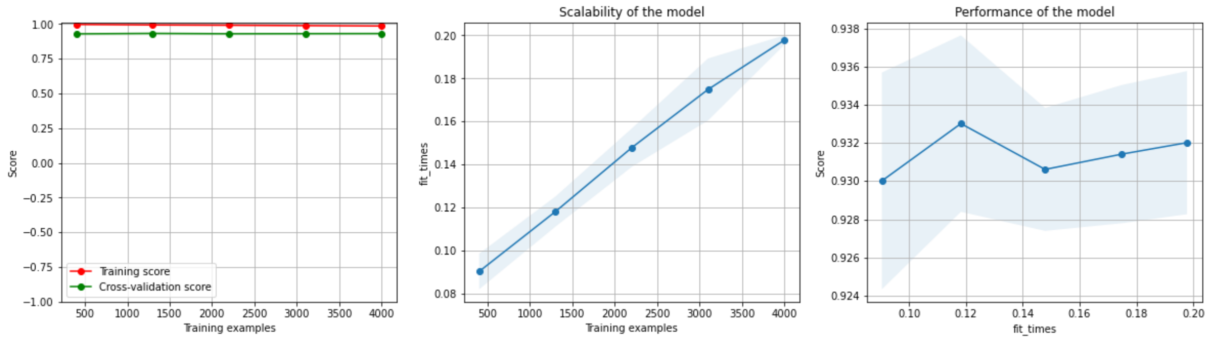
In [34]:
```
X_Most_Imp_Features.head(2)
```

Out[34]:

	Attack Diff	Speed diff
0	-6	-19
1	-39	0

In [35]:
```
plot_learning_curve(
    randomFlorestclf, "Random Forest Classifier", X_Most_Imp_Features
)
```

Out[35]: <module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packages/matplotlib/pyplot.py'>



In [36]:
```
X_Most_Imp_Features.head(3)
```

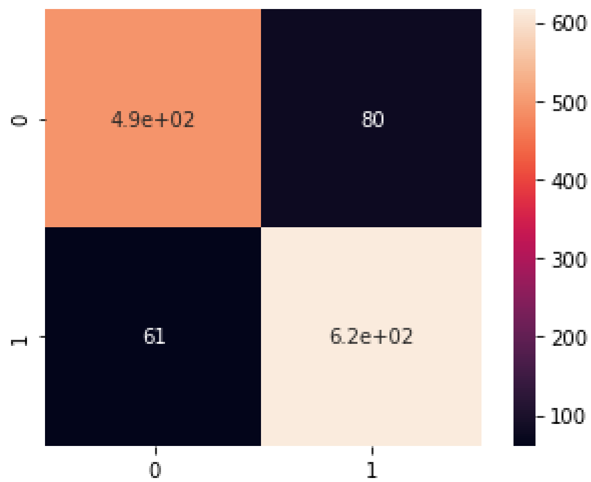
Out[36]:

	Attack Diff	Speed diff
0	-6	-19
1	-39	0
2	-35	0

In [37]:
```
# Logistic Regression difference between most imp features
# Attack Diff Speed Dif
clf = LogisticRegression(C=100,max_iter=10000)
clf.fit(X_train, y_train)
y_pred_log_reg = clf.predict(X_test)
acc_log_reg = round( clf.score(X_test, y_test) * 100, 2)
print(str(acc_log_reg) + ' percent')
sns.heatmap(confusion_matrix(y_test,y_pred_log_reg), square=True, anr
```

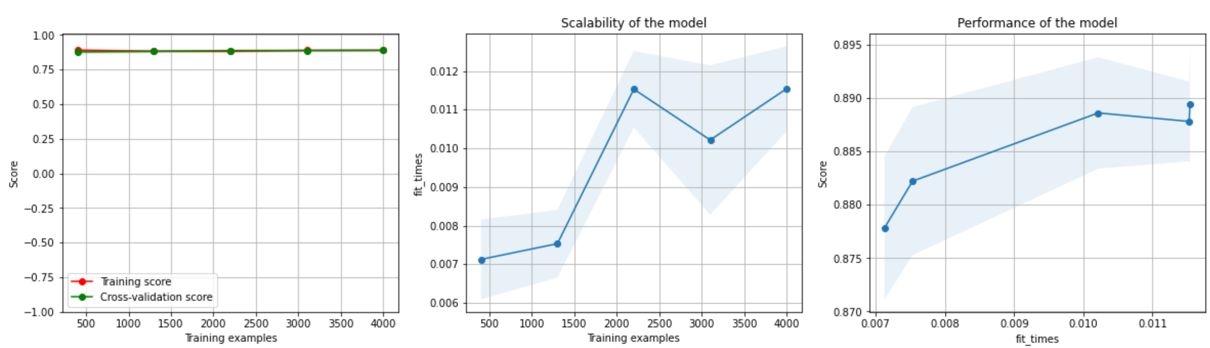
```
88.73 percent
```

Out[37]:
```
<AxesSubplot:>
```



In [38]:
```
plot_learning_curve(
    clf, "Logistic Regression", X_Most_Imp_Features, y,ylim=(-1, 1.01
)
```

Out[38]:
```
<module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packa
ges/matplotlib/pyplot.py'>
```



First approach

Pokemon stats in the same row types features encoded as a mapping

In [39]:
```
data = data_with_types.drop(columns=["Win", "Legendary", "Generation"
```

In [40]:
```
data.head(3)
```

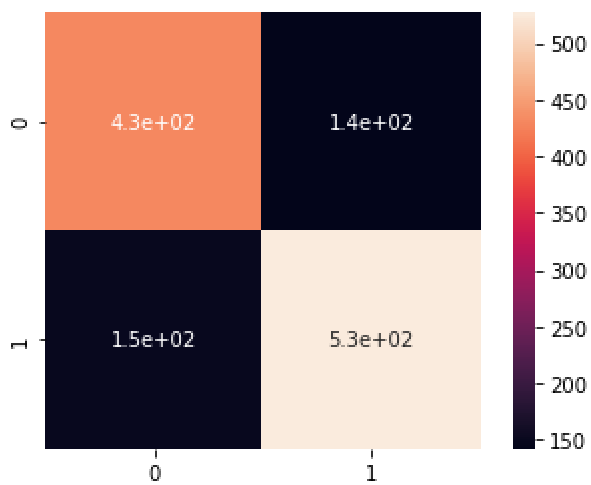
Out[40]:

	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	S D
0	10	13	50	64	50	45	50	41	17	16	70	70	40	60	
1	17	12	91	90	72	90	129	108	10	12	91	129	90	72	
2	15	9	55	40	85	80	105	40	2	0	75	75	75	125	

In [41]:
```
X = data.values[:, :-1].astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0

clf = LogisticRegression(C=0.1,max_iter=10000)
clf.fit(X_train, y_train)
y_pred_log_reg = clf.predict(X_test)
acc_log_reg = round( clf.score(X_test, y_test) * 100, 2)
print(str(acc_log_reg) + ' percent')
sns.heatmap(confusion_matrix(y_test,y_pred_log_reg), square=True, ann
```

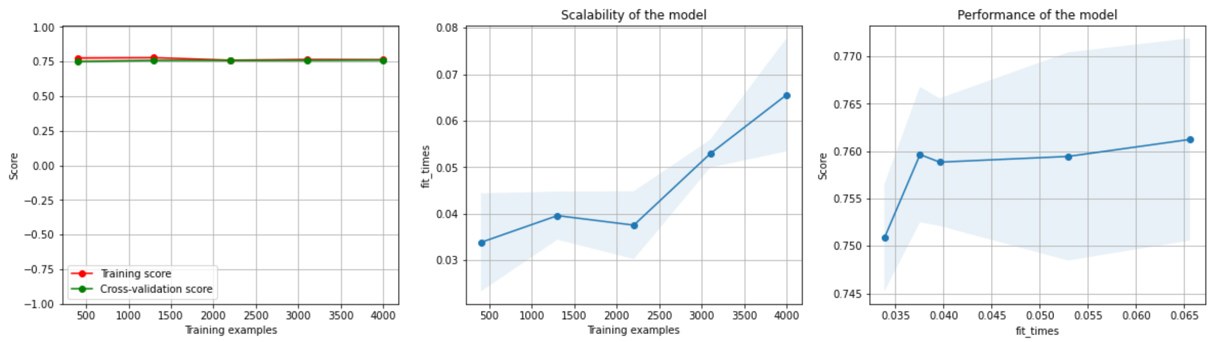
```
76.66 percent
```

Out[41]: <AxesSubplot:>



In [42]:
```
plot_learning_curve(
    clf, "Logistic Regression", X, y,ylim=(-1, 1.01), cv=5, n_jobs=4
)
```

Out[42]:
```
<module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packages/matplotlib/pyplot.py'>
```



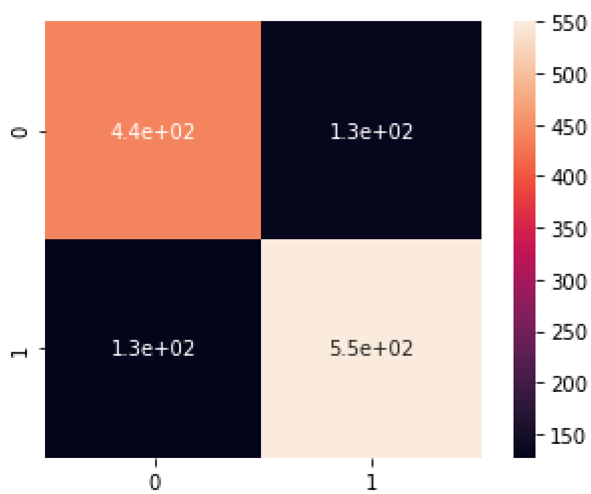
In [43]:
```
print(X[0])
```

```
[10 13 50 64 50 45 50 41 17 16 70 70 40 60 40]
```

In [44]:
```
# Random forest Classifier with every stat of both pokemons including
clf = RandomForestClassifier(n_estimators=100)
model = clf.fit(X_train,y_train)
pred = model.predict(X_test)
# print('Accuracy of {}:'.format(name), accuracy_score(pred, y_test))
print('Accuracy :', accuracy_score(pred, y_test))
print(classification_report(y_test, pred))
sns.heatmap(confusion_matrix(y_test, pred), square=True, annot=True)
```

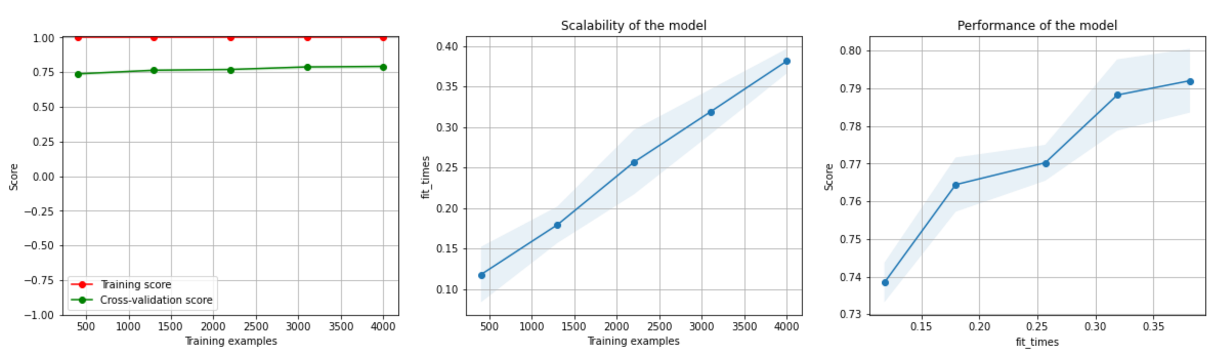
```
Accuracy : 0.7929656274980016
              precision    recall  f1-score   support

           0       0.78      0.77      0.77       574
           1       0.81      0.81      0.81       677

    accuracy                           0.79      1251
   macro avg       0.79      0.79      0.79      1251
weighted avg       0.79      0.79      0.79      1251
```

Out[44]: <AxesSubplot:>



In [45]:
```
plot_learning_curve(
    clf, "Random Forest Classifier", X, y,ylim=(-1, 1.01), cv=5, n_jo
)
```

Out[45]: <module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packages/matplotlib/pyplot.py'>

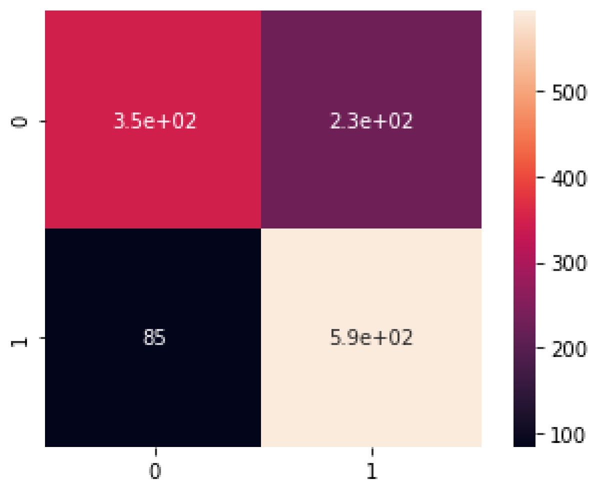


In [46]:

```
# MLP classifier with every stat as well as types map
clf = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(10
model = clf.fit(X_train, y_train)
pred = model.predict(X_test)
print('Accuracy :', accuracy_score(pred, y_test))
print(classification_report(y_test, pred))
sns.heatmap(confusion_matrix(y_test, pred), square=True, annot=True)
```

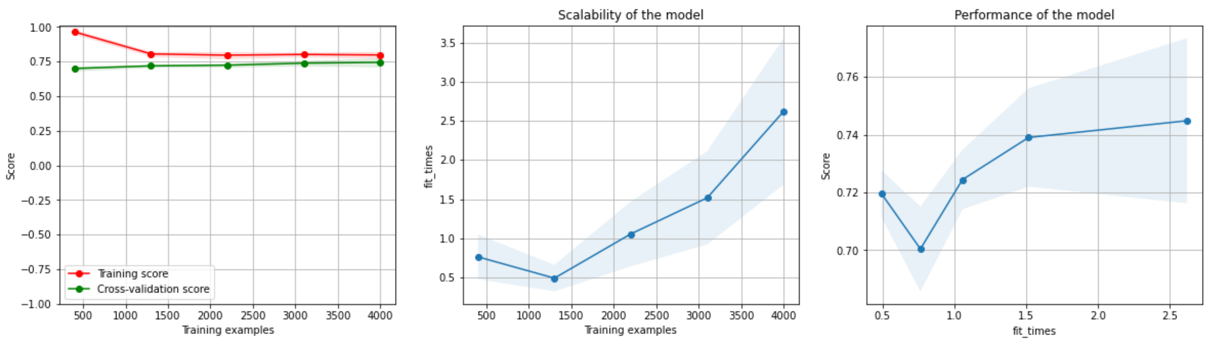
```
Accuracy : 0.7513988808952837
              precision    recall  f1-score   support

           0       0.80      0.61      0.69       574
           1       0.72      0.87      0.79       677

    accuracy                           0.75      1251
   macro avg       0.76      0.74      0.74      1251
weighted avg       0.76      0.75      0.75      1251
```

Out[46]: <AxesSubplot:>



In [47]:
```python
plot_learning_curve(
    clf, "MLP Classifier", X, y,ylim=(-1, 1.01), cv=5, n_jobs=4
)
```

Out[47]:
```
<module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packages/matplotlib/pyplot.py'>
```



All base stats normally appended to the data types as one hot encoding

In [48]:

```
# limit to categorical data using df.select_dtypes()
data_one_hot_encoding = pd.DataFrame( data_one_hot_encoding,columns=[
                                                                     'Defense', 'Sp. At
                                                                     'Generation', 'Leg
                                                                     'HP', 'Attack',
                                                                     'Defense', 'Sp. At
                                                                     'Generation', 'Leg
save = data_one_hot_encoding.copy()

data_one_hot_encoding = data_one_hot_encoding.astype({'HP' : 'int32',
                                                        'Defense': 'int
                                                        'Sp. Def' : 'in
                                                        'Generation': '
print(data_one_hot_encoding.dtypes)
X_objects = data_one_hot_encoding.select_dtypes(include=[object])
X_objects.head(3)
```

```
Type 1         object
Type 2         object
HP              int32
Attack          int32
Defense         int32
Sp. Atk         int32
Sp. Def         int32
Speed           int32
Generation      int32
Legendary       int32
Type 1         object
Type 2         object
HP              int32
Attack          int32
Defense         int32
Sp. Atk         int32
Sp. Def         int32
Speed           int32
Generation      int32
Legendary       int32
Winner          int32
dtype: object
```

Out[48]:

	Type 1	Type 2	Type 1	Type 2
0	Rock	Ground	Grass	Dark
1	Grass	Fighting	Rock	Fighting
2	Fairy	Flying	Psychic	NA

In [49]:

```

# 1. INSTANTIATE
# encode labels with value between 0 and n_classes-1.
le = preprocessing.LabelEncoder()

# 2/3. FIT AND TRANSFORM
# use df.apply() to apply le.fit_transform to all columns
X_objects_fitted = X_objects.apply(le.fit_transform)
X_objects_fitted.head()

# limit to categorical data using df.select_dtypes()
enc = preprocessing.OneHotEncoder()

# 2. FIT
enc.fit(X_objects_fitted)

# 3. Transform
onehotlabels = enc.transform(X_objects_fitted).toarray()
onehotlabels.shape
panda_Hot = pd.DataFrame(onehotlabels)
X_data_one_Hot = data_one_hot_encoding.drop(columns=["Winner", "Type

X_data_one_Hot = X_data_one_Hot.join(panda_Hot)
X_data_one_Hot.head()
```

Out[49]:

	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	HP	Attack	...	64	65	66
0	50	64	50	45	50	41	2	0	70	70	...	0.0	0.0	0.0
1	91	90	72	90	129	108	5	1	91	129	...	0.0	0.0	0.0
2	55	40	85	80	105	40	2	0	75	75	...	0.0	0.0	0.0
3	40	40	40	70	40	20	2	0	77	120	...	0.0	0.0	0.0
4	70	60	125	115	70	55	1	0	20	10	...	0.0	0.0	0.0

5 rows × 90 columns

In [50]:

```
X_train, X_test, y_train, y_test = train_test_split(X_data_one_Hot, y
print(X_train.shape)
print(y_train.shape)
```

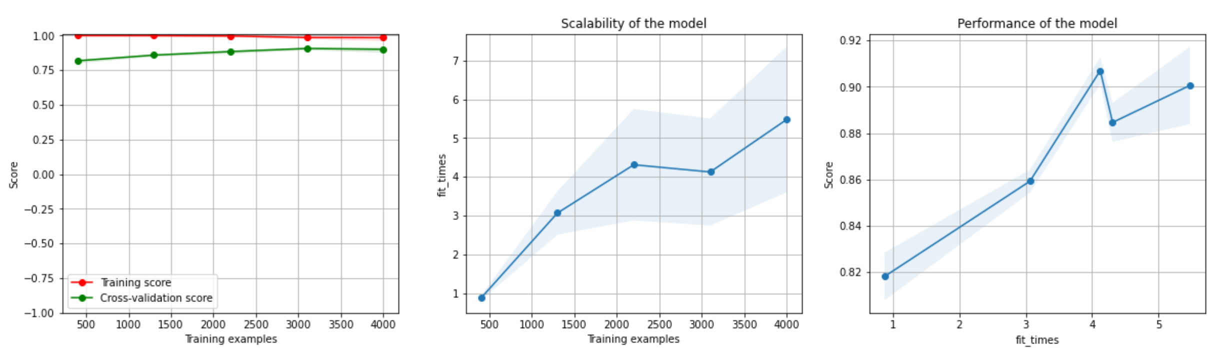
```
(3750, 90)
(3750,)
```

In [51]:
```
#this is bad3!
clf = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(10

y = data_one_hot_encoding.Winner
print(y.shape)

model = clf.fit(X_train, y_train)
pred = model.predict(X_test)
print(set(pred))
print(set(y_test))
# print('Accuracy of {}:'.format(name), accuracy_score(pred, y_test))
print('Accuracy :', accuracy_score(pred, y_test))
print(classification_report(y_test, pred))
```

```
(5001,)
{0, 1}
{0, 1}
Accuracy : 0.9072741806554756
              precision    recall  f1-score   support

           0       0.92      0.88      0.90       574
           1       0.90      0.93      0.92       677

    accuracy                           0.91      1251
   macro avg       0.91      0.91      0.91      1251
weighted avg       0.91      0.91      0.91      1251
```

In [52]:
```
plot_learning_curve(
    clf, "MLP Classifier", X_data_one_Hot, y,ylim=(-1, 1.01), cv=5, n
)
```

Out[52]:
```
<module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packa
ges/matplotlib/pyplot.py'>
```

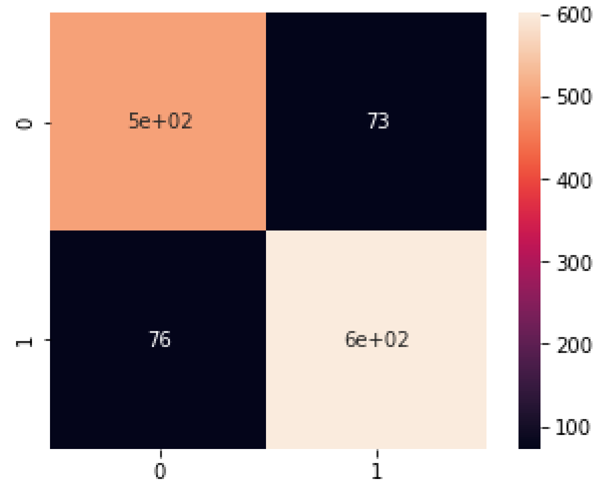


In [53]:

```
# Random Forest Classifier difference between most imp features
# Attack Diff Speed Dif
clf = RandomForestClassifier(n_estimators=100)
model = clf.fit(X_train,y_train)

pred = model.predict(X_test)
# print('Accuracy of {}:'.format(name), accuracy_score(pred, y_test))
print('Accuracy :', accuracy_score(pred, y_test))
print(classification_report(y_test, pred))
sns.heatmap(confusion_matrix(y_test, pred), square=True, annot=True)
```

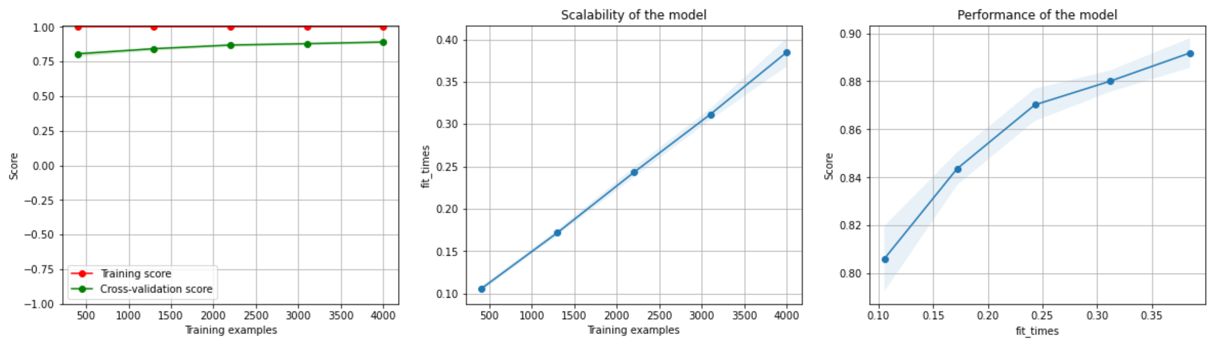
```
Accuracy : 0.8808952837729817
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       574
           1       0.89      0.89      0.89       677

    accuracy                           0.88      1251
   macro avg       0.88      0.88      0.88      1251
weighted avg       0.88      0.88      0.88      1251
```

Out[53]: <AxesSubplot:>



In [54]:
```
plot_learning_curve(
    clf, "Random Forest Classifier", X_data_one_Hot, y,ylim=(-1, 1.01
)
```

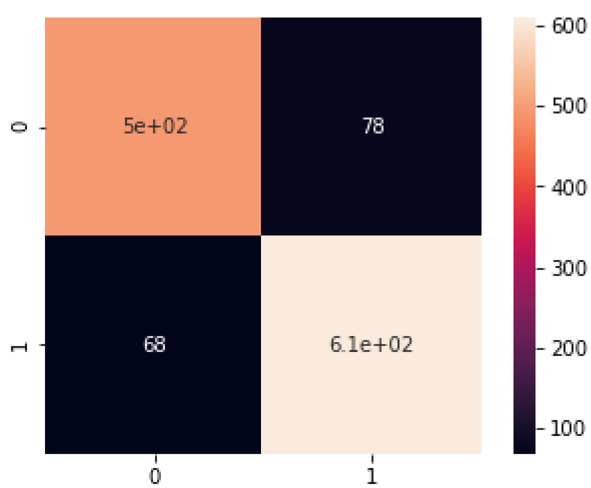
Out[54]: <module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packages/matplotlib/pyplot.py'>



In [55]:
```
clf = LogisticRegression(C=0.1,max_iter=10000)
clf.fit(X_train, y_train)
y_pred_log_reg = clf.predict(X_test)
acc_log_reg = round( clf.score(X_test, y_test) * 100, 2)
print(str(acc_log_reg) + ' percent')
sns.heatmap(confusion_matrix(y_test,y_pred_log_reg ), square=True, ar
```

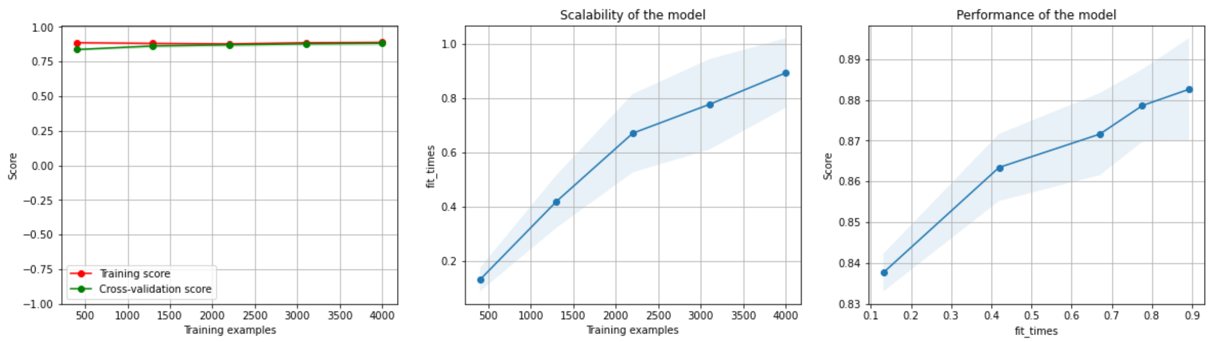
88.33 percent

Out[55]: <AxesSubplot:>



In [56]:
```
plot_learning_curve(
    clf, "Logistic Regression", X_data_one_Hot, y,ylim=(-1, 1.01), cv
)
```

Out[56]:
```
<module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packa
ges/matplotlib/pyplot.py'>
```



Third Approach

Use only attack speed and types as a map

In [57]:
```
X = data_with_types[["Type 1", "Type 2", "Attack", "Speed"]]
X.head(3)
```

Out[57]:

	Type 1	Type 1	Type 2	Type 2	Attack	Attack	Speed	Speed
0	10	17	13	16	64	70	41	60
1	17	10	12	12	90	129	108	108
2	15	2	9	0	40	75	40	40

In [58]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0
print(X_train.shape)
print(y_train.shape)
```

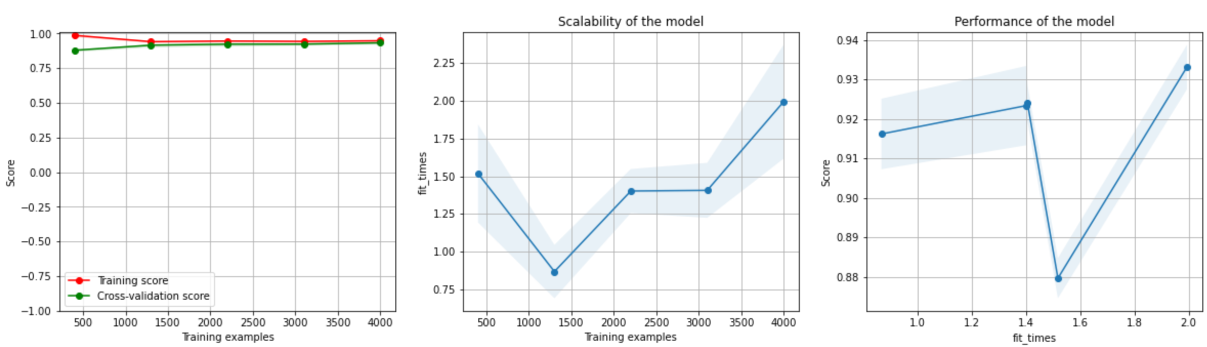
```
(3750, 8)
(3750,)
```

In [59]:
```
#this is bad3!
clf = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(10

y = data_one_hot_encoding.Winner
print(y.shape)

model = clf.fit(X_train, y_train)
pred = model.predict(X_test)
print(set(pred))
print(set(y_test))
# print('Accuracy of {}:'.format(name), accuracy_score(pred, y_test))
print('Accuracy :', accuracy_score(pred, y_test))
print(classification_report(y_test, pred))
```

```
(5001,)
{0, 1}
{0, 1}
Accuracy : 0.9232613908872902
              precision    recall  f1-score   support

           0       0.95      0.88      0.91       574
           1       0.90      0.96      0.93       677

    accuracy                           0.92      1251
   macro avg       0.93      0.92      0.92      1251
weighted avg       0.92      0.92      0.92      1251
```

In [60]:
```
plot_learning_curve(
    clf, "MLP Classifier", X, y,ylim=(-1, 1.01), cv=5, n_jobs=4
)
```

Out[60]:
```
<module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packa
ges/matplotlib/pyplot.py'>
```

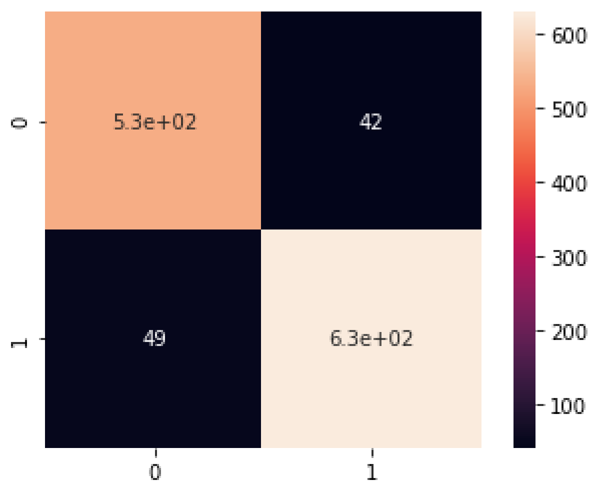


In [61]:

```
# Random Forest Classifier difference between most imp features
# Attack Diff Speed Dif
clf = RandomForestClassifier(n_estimators=100)
model = clf.fit(X_train,y_train)

pred = model.predict(X_test)
# print('Accuracy of {}:'.format(name), accuracy_score(pred, y_test))
print('Accuracy :', accuracy_score(pred, y_test))
print(classification_report(y_test, pred))
sns.heatmap(confusion_matrix(y_test, pred), square=True, annot=True)
```

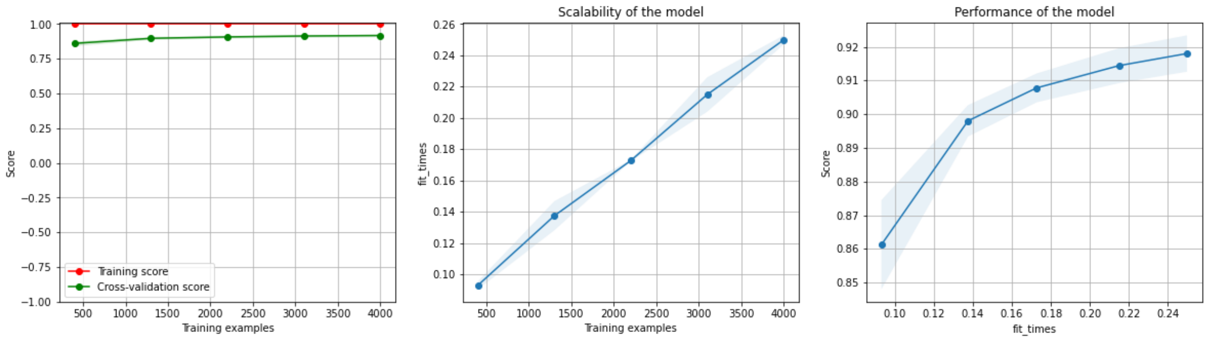
```
Accuracy : 0.9272581934452439
              precision    recall  f1-score   support

           0       0.92      0.93      0.92       574
           1       0.94      0.93      0.93       677

    accuracy                           0.93      1251
   macro avg       0.93      0.93      0.93      1251
weighted avg       0.93      0.93      0.93      1251
```

Out[61]: <AxesSubplot:>



In [62]:
```
plot_learning_curve(
    clf, "Random Forest Classifier", X, y,ylim=(-1, 1.01), cv=5, n_jo
)
```

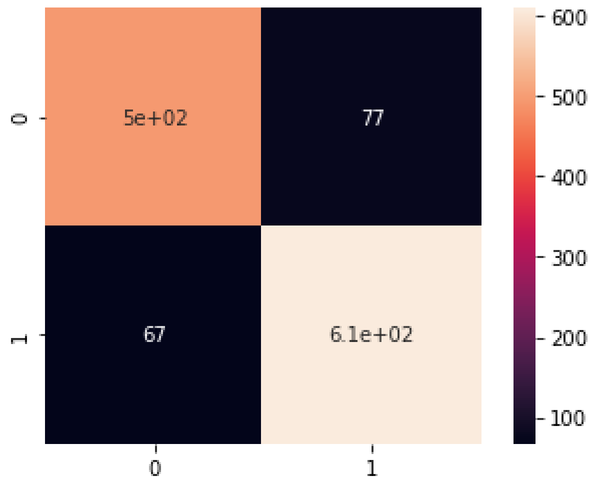
Out[62]: <module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packages/matplotlib/pyplot.py'>



In [63]:
```

clf = LogisticRegression(C=0.1,max_iter=10000)
clf.fit(X_train, y_train)
y_pred_log_reg = clf.predict(X_test)
acc_log_reg = round( clf.score(X_test, y_test) * 100, 2)
print(str(acc_log_reg) + ' percent')
sns.heatmap(confusion_matrix(y_test,y_pred_log_reg  ), square=True, a
```

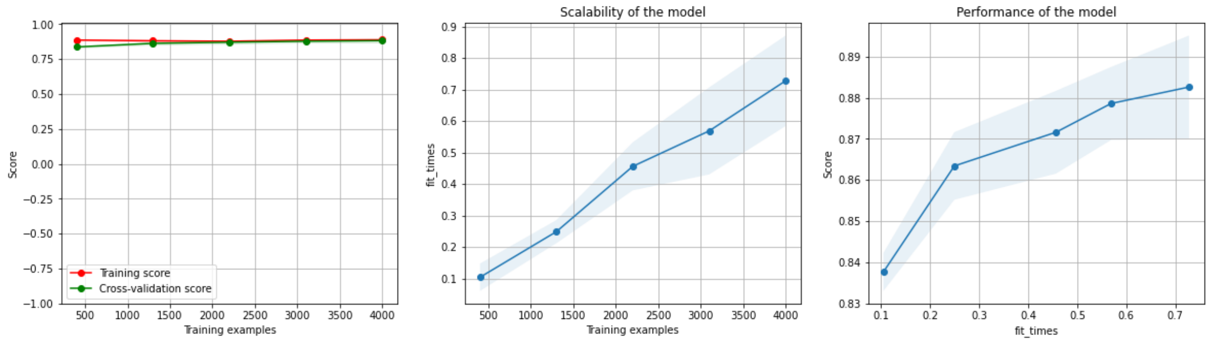
88.49 percent

Out[63]: <AxesSubplot:>



In [64]:
```
plot_learning_curve(
    clf, "Logistic Regression", X_data_one_Hot, y,ylim=(-1, 1.01), cv
)
```

Out[64]:
```
<module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packa
ges/matplotlib/pyplot.py'>
```



Use difference stats and types mapped to integers

In [65]:
```
X_Most_Imp_Features = X_Most_Imp_Features.join(data_with_types[["Type
```

In [66]:
```
X_Most_Imp_Features.head(3)
```

Out[66]:

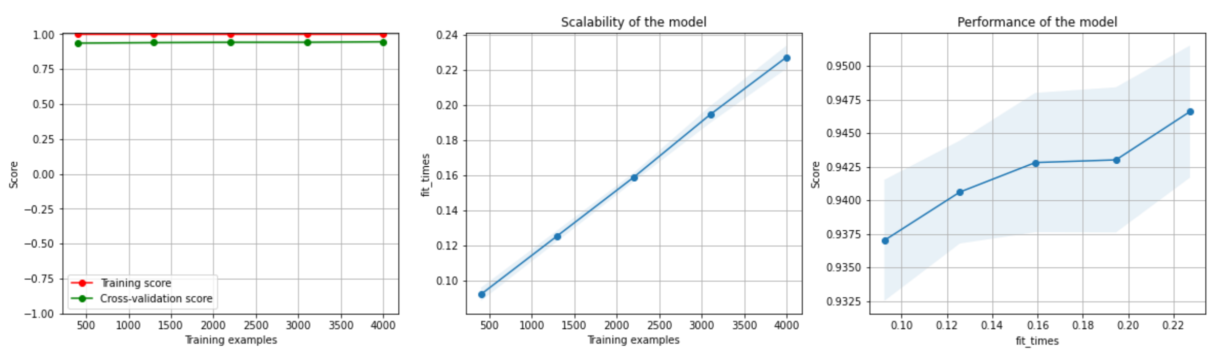
	Attack Diff	Speed diff	Type 1	Type 1	Type 2	Type 2
0	-6	-19	10	17	13	16
1	-39	0	17	10	12	12
2	-35	0	15	2	9	0

In [83]:
```
X_train, X_test, y_train, y_test = train_test_split(X_Most_Imp_Featur
randomFlorestclf = RandomForestClassifier(n_estimators=100)
model = randomFlorestclf.fit(X_train,y_train)
pred = model.predict(X_test)
# print('Accuracy of {}:'.format(name), accuracy_score(pred, y_test))
print('Accuracy :', accuracy_score(pred, y_test))
print(classification_report(y_test, pred))
```

```
Accuracy : 0.9496402877697842
              precision    recall  f1-score   support

           0       0.94      0.95      0.95       574
           1       0.96      0.95      0.95       677

    accuracy                           0.95      1251
   macro avg       0.95      0.95      0.95      1251
weighted avg       0.95      0.95      0.95      1251
```

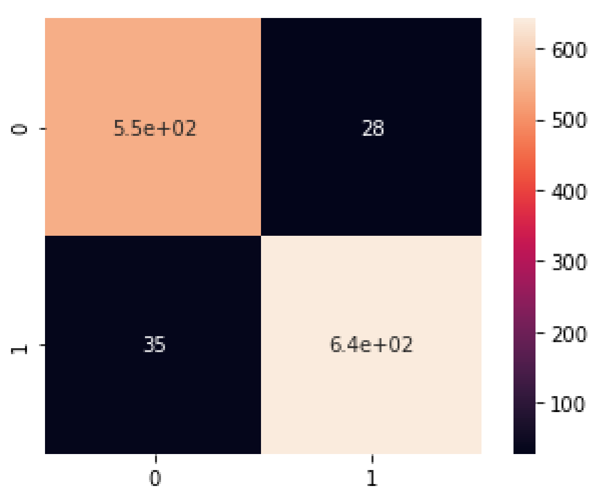
In [84]:
```
plot_learning_curve(
    randomFlorestclf, "Random Forest Classifier", X_Most_Imp_Features
)
```

Out[84]: <module 'matplotlib.pyplot' from '/opt/anaconda/lib/python3.8/site-packages/matplotlib/pyplot.py'>



In [85]:
```
sns.heatmap(confusion_matrix(y_test, pred), square=True, annot=True)
```

Out[85]: <AxesSubplot:>



In [70]:

```python

train_features, test_features, train_labels, test_labels = train_test

```

In [ ]:

```python
from sklearn.model_selection import RandomizedSearchCV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000,
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestClassifier(random_state = 42)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cor
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=ranc
                              n_iter = 10000, scoring='neg_mean_absol
                              cv = 3, verbose=2, random_state=42, n_j
                              return_train_score=True)



# Fit the random search model
rf_random.fit(train_features , train_labels);
```

Evaluate the Default Model

In [73]:
```
base_model = RandomForestClassifier(n_estimators = 100, random_state
base_model.fit(train_features, train_labels)
pred = base_model.predict(test_features)

base_acc = accuracy_score(pred, test_labels)

# print('Accuracy of {}:'.format(name), accuracy_score(pred, y_test))
print('Accuracy :', accuracy_score(pred, test_labels))
print(classification_report(pred, test_labels))

```

```
Accuracy : 0.947242206235012
              precision    recall  f1-score   support

           0       0.95      0.94      0.94       580
           1       0.95      0.96      0.95       671

    accuracy                           0.95      1251
   macro avg       0.95      0.95      0.95      1251
weighted avg       0.95      0.95      0.95      1251
```

Evaluate the Best Random Search Model

In [ ]:
```
best_random = rf_random.best_estimator_

pred = best_random.predict(test_features)

# print('Accuracy of {}:'.format(name), accuracy_score(pred, y_test))
random_acc = accuracy_score(pred, test_labels)
print('Accuracy :', accuracy_score(pred, test_labels))
print(classification_report(pred, test_labels))

print('Improvement of {:0.2f}%.'.format( 100 * (random_acc- base_acc)
```

Introducing new data to see how the final model behaves

In [81]:

```python

test_labels = extra_data_pd_frame ["Win"].astype(int).values

X_new =  extra_data_pd_frame[["Attack Diff", "Speed diff"]]
X_new = X_new.join(extra_data_with_types[["Type 1", "Type 2"]])

pred = randomFlorestclf.predict(X_new)

# print('Accuracy of {}:'.format(name), accuracy_score(pred, y_test))
random_acc = accuracy_score(pred, test_labels)
print('Accuracy :', accuracy_score(pred, test_labels))
print(classification_report(pred, test_labels))






sns.heatmap(confusion_matrix(test_labels, pred), annot=True)
```

```
Accuracy : 0.9441776710684273
              precision    recall  f1-score   support

           0       0.95      0.93      0.94      2393
           1       0.94      0.95      0.95      2605

    accuracy                           0.94      4998
   macro avg       0.94      0.94      0.94      4998
weighted avg       0.94      0.94      0.94      4998
```

Out[81]: <AxesSubplot:>